



SUBMISSION OF WRITTEN WORK

Class code: BFST

Name of course: FØRSTEÅRSPROJEKT

Course manager: TROELS BJERRE SØRENSEN

Course e-portfolio: <https://giddy.itu.dk/GruppeB>

Thesis or project title: JESTA

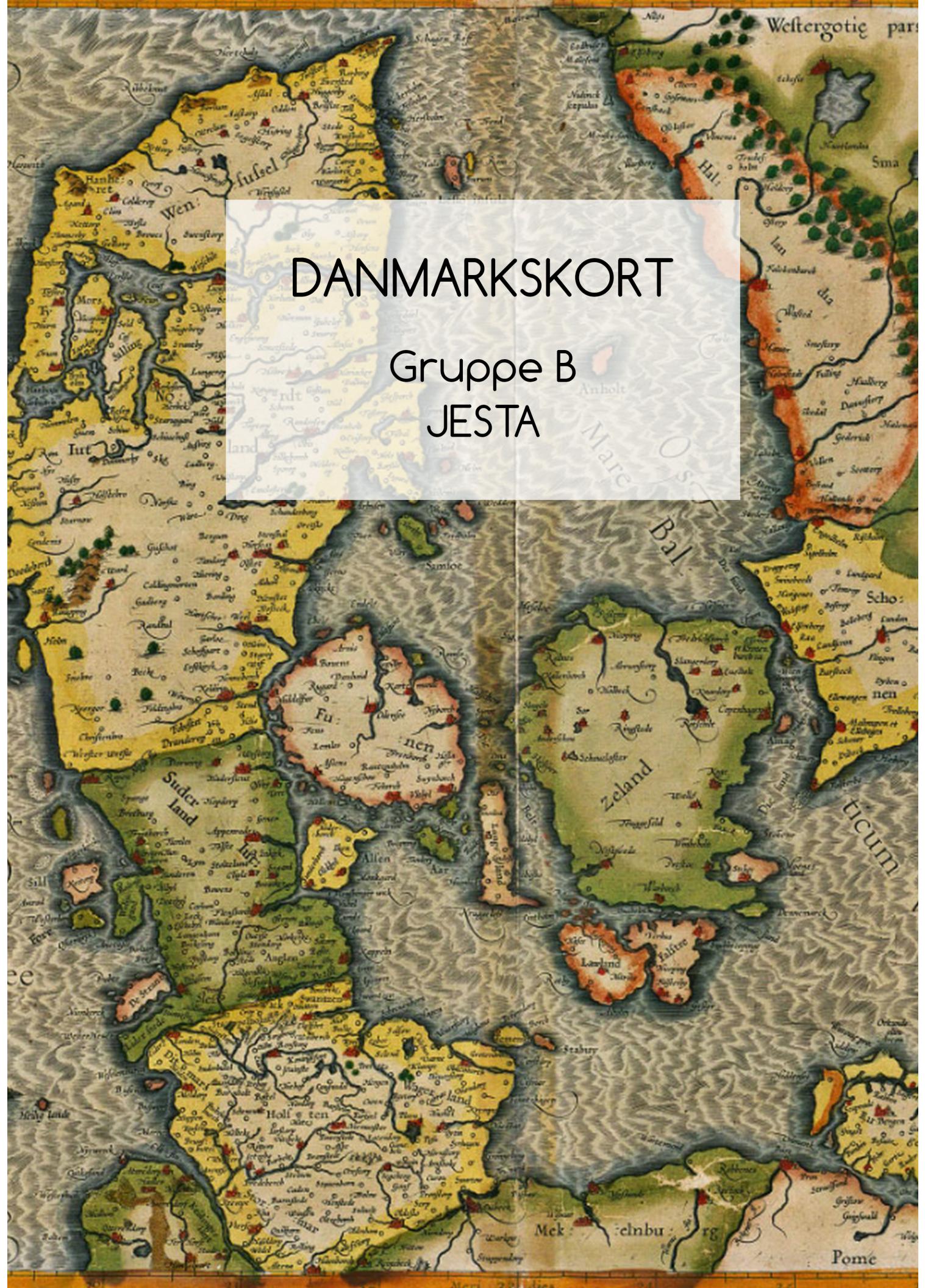
Supervisor: RASMUS GREVE

Full Name:

1. <u>AMANDA ENGEL THILO</u>	Birthdate (dd/mm-yyyy): <u>21/03-1993</u>	E-mail: <u>AENT@itu.dk</u>
2. <u>EMIL REFGAARD MIDDLEBOE</u>	Birthdate (dd/mm-yyyy): <u>02/09-1994</u>	E-mail: <u>EREM@itu.dk</u>
3. <u>JAKOB PÆRREGAARD HOLM</u>	Birthdate (dd/mm-yyyy): <u>25/02-1995</u>	E-mail: <u>JAPH@itu.dk</u>
4. <u>STEFAN BJØRN PETURSSON</u>	Birthdate (dd/mm-yyyy): <u>15/10-1994</u>	E-mail: <u>SBJR@itu.dk</u>
5. <u>THOR V.A.N OLESEN</u>	Birthdate (dd/mm-yyyy): <u>14/02-1995</u>	E-mail: <u>TVAO@itu.dk</u>
6. _____	_____	_____@itu.dk
7. _____	_____	_____@itu.dk

DANMARKSKORT

Gruppe B
JESTA



Denne rapport er udarbejdet i Maj 2015 gennem et 12 ugers projekt på IT-Universitetet i København under vejledning af Troels Bjerre Sørensen og TA Rasmus Greve. I projektet udviklede vi et softwaresystem til kortlægning af Danmark samt rutesøgning skrevet i Java. Programmet gør brug af open street maps [Maps] og skal kunne køre hele Danmarkskortet.

Indhold

1 Baggrund	7
1.1 Problemområde	7
1.2 Krav til produktet	8
2 Overvejelser	9
2.1 Clean Code	9
2.1.1 Konventioner	9
2.1.2 Navnekonventioner	9
2.1.3 Metoder	10
2.1.4 Klasser	10
2.1.5 Variabler	10
2.1.6 Kommentarer	11
2.2 JavaFX vs Swing	11
2.3 Opsummering	11
3 Analyse og design af brugergrænseflade	12
3.1 Designfasen	12
3.2 Virtuelle vinduer	12
3.3 Endeligt designvalg	13
3.4 Farveblindhed	13
3.5 Kortvisning	14
3.6 Opsummering	14
4 Teknisk Analyse	15
4.1 MVC designmønster	15
4.2 Starter-klassen	16
4.3 Implementering af brugergrænseflade	16
4.3.1 MapView klassen	16
4.3.2 Canvas klassen	17
4.3.3 AffineTransform	18
4.4 OSMHandler klassen	19
4.4.1 Formål	19
4.4.2 Virkemåde	19
4.4.3 Vejtyper	20

4.4.4	Kystlinjer	20
4.4.5	Veje	21
4.5	Serializable	21
4.5.1	Definition	21
4.5.2	API	22
4.5.3	Kortdata	22
4.5.4	Implementation	22
4.5.5	Fordele og ulemper	22
4.6	Datastrukturer	23
4.6.1	Quadtræ til kortdata	23
4.6.2	KD-træ til adresser	26
4.6.3	Dijkstra til rutesøgning	28
4.6.4	Rutevejledning	31
4.7	Addressesøgning	31
4.7.1	Glazed List	31
4.7.2	Trie	32
4.7.3	SwingX	32
4.7.4	Udfordringer og særlige valg af opsætning	32
4.8	Opsummering	32
5	Teknisk beskrivelse	33
5.1	Struktur - UML diagrams	33
5.2	Strukturdigrammer	33
5.2.1	Pakkediagrammer	33
5.2.2	Klassediagrammer	33
6	Testing	35
6.1	Black Box	35
6.2	Address Parser	35
6.2.1	Definitionen på en adresse	36
6.2.2	Test cases	36
6.2.3	Ækvivalensklasser	36
6.2.4	Vejnavn	36
6.2.5	Husnummer	36
6.3	Andre testklasser	37
6.3.1	TestTST	37
6.3.2	OSMHandlerTest	37
6.3.3	CalculatorTest	37
6.3.4	DataCollectionModelTest	37
6.3.5	DijkstraTest	37
6.4	TestMaps	37
6.5	Test Fixtures	38
6.6	White Box	38

6.6.1	Quadtræ - insertion metode	38
6.7	Opsummering	40
7	Brugerguide	41
8	Konklusion	42
8.1	Videre udvikling	42
8.1.1	Høj kobling	42
8.1.2	Rutevejledning	42
8.1.3	AddressParser	43
8.1.4	Klassestruktur	43
8.2	Konklusion	43
9	Reflektion over projektet	44
Figurer		45
A	Bilag	48
A.1	Kildekode	48
A.1.1	Clean Code - Eksempel 1	48
A.1.2	Clean Code - Eksempel 2	49
A.1.3	Clean Code - Eksempel 3	49
A.1.4	Canvas klassen - Tags Array og tegneproces	50
A.1.5	OSMHandler - Eksempel 1	51
A.1.6	OSMHandler - Eksempel 2	51
A.1.7	AddressParser - Eksempel 1	52
A.1.8	Serializable - Eksempel 1	52
A.1.9	Serializable - Eksempel 2	53
A.2	Whitebox test af insertion i quadtræ	54
A.2.1	Coverage tabel over alle kontrolpunkter - del 1	54
A.2.2	Coverage tabel over alle kontrolpunkter - del 2	55
A.2.3	Input datasæt med forventede og aktuelle resultater	56
A.3	Gruppekontrakt	57
A.3.1	Medlemmer	57
A.3.2	Retningslinjer	57
A.3.3	Principper & arbejdsnormer	58
A.3.4	Mødestruktur	60
A.3.5	Mødetidspunkter	61
A.3.6	Koderegler	61
A.4	Logbog	62
A.4.1	25-02-15	62
A.4.2	28/02-2015	63
A.4.3	2/3 - 2015	64
A.4.4	3/3 - 2015	65

A.5	Timelog	84
A.6	Change-log	90
A.6.1	v0.1	90
A.6.2	v0.2	90
A.6.3	v0.3	91
A.6.4	v0.4	91
A.6.5	v0.5	92
A.6.6	v0.6	93
A.6.7	v0.7	94
A.6.8	v0.8	94
A.6.9	v0.9	95
A.6.10	v1.0	95
A.6.11	Feature Freeze	96

Kapitel 1

Baggrund

1.1 Problemområde

I forbindelse med førsteårsprojektet, er vi blevet bedt om at udvikle et program, der kortlægger hele Danmarkskortet og understøtter gængse funktionaliteter indenfor rutevejledning og navigation. For at imødekomme dette sammen med de konkrete krav nedenunder, bliver man nødt til at undersøge de problemer, der opstår indenfor programmets domæne. En af hovedudfordringerne ved programmet er relateret til programmets ressourcekrav. Programmet skal rendere hele Danmarkskortet dynamisk og give brugeren et overblik over relevant geografisk information på alle zoomniveauer. Dette indbefatter nødvendigvis brugen af effektive datastrukturerer til at holde på den omfattende kortdata og kræver samtidig, at man foretager nogle bevidste valg, om hvad der skal tegnes på de forskellige zoomniveauer. Til slut er det nødvendigt at bruge effektive algoritmer til at udregne en rute mellem to punkter på kortet uden, at alle veje skal gennemløbes først, da programmet ellers vil køre for langsomt til, at det kan opfylde de vigtigste brugerkrav. Alt dette er mundet ud i et objekt orienteret design, der forsøger at modellere problemdomænet, hvor programmets objekter tilsvarer objekter i domænet.

1.2 Krav til produktet

Det endelige produkt skal opfylde følgende krav:

- Tillade brugeren at specificere en fil med kortdata til brug i programmet. Som minimum skal programmet kunne læse en zippet .OSM fil.
- Tillade brugeren at gemme og læse det nuværende kort i et binært format - dette skal dokumenteres i rapporten.
- Inkludere en standard binær fil implementeret i en jar fil. Således at standard filen bliver læst, hvis der ikke gives anden .OSM fil.
- Tegne alle veje på det pågældende kort. Vejene skal tegnes i forskellige farver.
- Tegne kartografiske elementer.
- Tillade brugeren at fokusere på specifikke områder på kortet; enten ved at tegne en firkant og zoome ind, eller ved først at zoome og derefter panorere.
- Brugergrænsefladen skal ændre sin størrelse dynamisk, når man trækker i vinduet.
- Vise vejnavne; enten når musen holdes over en given vej eller i en separat statusbar.
- Tillade brugeren at søge efter adresse - indtastet som en enkelt streng. Resultatet skal vises på kortet. Ukendt input skal håndteres, fx ved at give brugeren en liste med mulige matches.
- Programmet skal kunne opgive den korteste vej mellem to punkter givet af brugeren.
- Tillade brugeren at vælge mellem om en given rute skal foregå på cykel, gåben eller i bil. Ruter i bil skal tage højde for fartgrænser ved udregning af rutens forventede rejsetid. Ruter for cykler og gående skal være de korteste og skal undgå motorveje.
- Programmet skal understøtte en brugervenlig måde at navigere rundt på kortet - enten via tastatur eller mus.
- Programmet skal kunne give en tekstbeskrivelse tilknyttet en given rute. Herunder skal der gives rutevejledning og instrukser til brugeren. Eksempelvis: ”Følg Rued Langgaardsvej i 50 m, derefter drej til højre ind på Røde Mellemvej.”
- Programmet skal indeholde en måde at vise zoom-level på. Fx via geometriske informationer.
- Tillade brugeren at skifte udseende på kortet i tilfælde af farveblindhed, hvor specielt rød-grønne farver vil fremstå anderledes og kan lede til forvirring om geografiske informationer på kortet.
- Programmet skal være hurtigt nok til, at det er fordelagtigt at bruge. Dette skal også gøre sig gældende for store datasæt såsom Danmarks kortet, der er i brug.

Kapitel 2

Overvejelser

Dette kapitel vil fokusere på de vigtigste overvejelser, som vi har gjort os før og under projektet. Herunder de retningslinjer vi har valgt at følge.

2.1 Clean Code

I forløbet er vi gennem TA's blevet introduceret til bogen 'Clean Code' af Robert C. Martin, der som navnet antyder, handler om at skrive kode, der er læsevenligt. Clean code har haft en stor betydning i vores projekt, hovedsagligt fordi man i en større gruppe, som vores har behov for nemt at kunne forstå, ændre og videreudvikle hinandens kode. Desuden er dårlig kode enormt dyrt, da man har sværere ved at vedligeholde programmet og finde fejl, som derfor kræver ekstra arbejde og tid (penge). Clean Code er i dette henseende en læserdrevne løsning, der producerer software, som er let at skrive, læse og vedligeholde. Derfor bør man også have i mente, at det bygger på en erkendelse om, at man ikke bare skriver koden for at få den til at virke på computeren, men snarere med henblik på at gøre den forståelig for mennesker også. En af fordelene ved clean code er, at man opnår en kildetekst, der er meget modulær og derfor nemmere at læse, forstå og teste. I forbindelse med dette er testing, refaktorering og TDD¹ kommet meget frem i søgeresultatet, da de er med til at højne kvaliteten af koden og vedligeholdelsen [TA's].

2.1.1 Konventioner

For at imødekomme Clean Code, har vi efterstræbt at følge nogle konventioner indenfor navngivning og opbygning af vores klasser, metoder og variabler. Herudover har vi haft meget fokus på at gøre koden selvdokumenterende og samtidig understøtte den med javadokumentationen. Målet har været at opnå et modulært design, hvor koden er nem at forstå, ændre og teste.

2.1.2 Navnekonventioner

Ved navngivningen af klasser, metoder og variabler er det vigtigt at tilkendegive, hvad formålet med dem er, og efterstræbe at beskrive intentionen bag. Vi har navngivet klasserne med navneord og metoderne med verber. Formålet har været at afsløre intentionen bag koden gennem navnene, så man ikke behøver at nærlæse implementeringen for at få en forståelse af programmets struktur.

¹Test Driven Development

Af samme grund har vi valgt at de klasser der har samme roller også har samme navne, fx *FileSaver*, *FileChooser* og *FileLoader*. Navnene antyder at deres funktionaliteter har med filer at gøre. Det er også vigtigt at angive en kontekst for klasserne og deres indbyrdes forhold. Dette er beskrevet øverst i en kommentar for hver af klasserne, så læseren har den nødvendige baggrundsviden til at forstå klassens ansvarsområde og relation til resten af programmet (se bilag: Clean code - Eksempel 1).

2.1.3 Metoder

For at opfylde Clean Code principperne har vi så vidt muligt sat alle metoder som hjælpefunktioner, der skal være korte, beskrivende og selvforklarende. Dette har vi opnået ved at sørge for at metoderne så vidt muligt kun varetager en funktion. Tilsammen udgør metoderne en sammenhængende entitet. Alt dette har vi benyttet os af til at refaktorere større dele af koden, hvor metoderne var for omfattende og dækkede flere funktioner samtidig. Helt konkret har vi kigget på vores oprindelige kode, og ledt efter funktionaliteter, der kunne separeres fra eksisterende metoder. Der er eksempelvis forskel på om man tilgår et objekt eller søger svar på spørgsmål om et objekt. Clean code - Eksempel 2 i bilag illustrerer to metoder, der bruges til at bygge knapperne til rutevejledning i programmets sidepanel. Metodernes formål udtrykkes tydeligt gennem navngivningen, der gør koden selvforklarende. Førstnævnte bygger to knapper og benytter en anden metode til at definere deres størrelser. Derved er to funktionaliteter separeret fra hinanden istedet for at blive kaldt i én metode, der håndterer flere arbejdsopgaver.

2.1.4 Klasser

Klasserne er ligesom metoder med til at organisere kode og give det en kontekst, som det skal forstås i. Med dette i mente har vi efterstræbt at afdække de forskellige ansvarsområder med klasser. Klasserne begrænses til ét ansvarsområde, ligesom metoderne begrænses til et abstraktionsniveau. Dette princip er også benævnt SRP². For at opnå dette har vi ligesom med metoder forsøgt at trække ansvarsområder ud af klasser, der håndterer flere ting på en gang. Ved at trække ansvarsområder ud i nye klasser, bliver kohäsionen (sammenhængskraften) reduceret i programmet, så tingene ikke afhænger gensidigt af hinanden. Et eksempel på dette er vores klasse *Calculator*, som udelukkende har udregner afstande mellem koordinatsæt.

2.1.5 Variabler

Navne på variabler bør være afslørende, meningsfulde og udtryksfulde. Navnene bør beskrive intentionen med variablerne og ikke mislede læseren. Meningsfulde navne er med til at gøre koden selvforklarende og gør behovet for dokumentation mindre. Eksempelvis er der stor forskel på om en liste er navngivet 'list1' eller 'shapeList'. Førstnævnte baserer sig på en høj abstraktion og er intetsigende om, hvad variablen bliver brugt til. Sidstnævnte forklarer en, at der er tale om en liste, som holder på 'shapes' - i dette tilfælde er der tale om de 'figurer' vi i sin tid brugte til at repræsentere forskellige dele af kortet (nu PathTags). Se Clean code - Eksempel 3 for et kodeudsnit med fokus på navngivning af variabler i programmet.

²Single Responsibility Principle

2.1.6 Kommentarer

Som nævnt før kan man gennem en god navngivning af klasser, metoder og variabler gøre koden selv-dokumenterende og selvforklarende. I bogen 'Clean Code' hævder forfatteren ligefrem, at kommentarer bliver brugt af programmører, der ikke er gode nok til at forklare, hvad de vil i kode. Dette har vi dog vurderet at være til ekstremen, og har konsekvent kommenteret det meste af vores kode af hensyn til flere grunde. Først og fremmest er det i sig selv en god øvelse, og det tvinger en til at forstå, hvad der sker i koden. Derudover er det en hjælp for hele gruppen, da vi som udgangspunkt ikke har de samme forudsætninger og vi ikke kan arbejde ligeligt på alt. Kommentarer er en hjælp til at følge op på ændringer løbende og sætte sig ind i andres metodik og tankegang. Vi har bestræbt os på at beskrive formålet bag vores kode for både klasserne generelt og metoderne specifikt. Dertil skal nævnes at alle offentlige metoder har fået tilføjet javaDoc kommentarer, så man kan læse hvad de gør, og hvordan de fungerer uden at skulle tilgå koden. De steder hvor der er behov for en kontekst, har vi prøvet at skabe en sammenhæng og relation til resten af koden. Eksempelvis havde vi tidligere beskrevet at vores 'DrawShape' klasse benytter subklasserne i Shapes pakken til at tegne forskellige dele af kortet.

2.2 JavaFX vs Swing

Valget mellem JavaFX og Swing til programmet blev overvejet nøje i starten af processen, da det udgør en bærende del af programmet. Først og fremmest er grafikken generelt bedre renderet i JavaFX end i Swing, fordi det kører over grafikkortet frem for processoren. Derudover erstatter JavaFX officielt Swing som Oracle's UI Library for Java. Samtidig er det nemt at style JavaFX applikationen [] agennem CSS. Hvad mere er at JavaFX har et plugin 'Scene Builder' som en drag-n-drop værktøj, hvilket gør nemt at komme hurtigt igang med at bygge applikationer og gør det nemt at ændre brugergrænsefladen hurtigt og effektivt. Slutteligt er det lettere at følge et MVC design, hvor JavaFX med FXML automatisk lægger op til et MVC design. Til gengæld optræder der en række komplikationer ved anvendelse af JavaFX til netop dette projekt. Swing har derimod en række fordele, som er tiltalende i netop dette projekt. Swing tegner et stort antal linjer langt hurtigere og anvender mindre hukommelse undervejs []. Der er desuden langt mere dokumentation i Swing end i JavaFx, hvilket gør det mere attraktivt at arbejde med, da løsninger på problemstillinger er nemmere at finde. Til slut kender TA's Swing langt bedre end JavaFX og vil have svært ved at hjælpe os, hvis vi havde valgt at bruge JavaFx [JavaFX].

2.3 Opsummering

Ideen om *Clean Code* har haft en gennemgående betydning i projektforløbet og slutproduktet. Vi har efterstræbt at udvikle et program med fundament i læsevenlig, overskuelig og selvforklarende kode vha. konventioner og kommentarer. Vores valg af Swing frem for JavaFX har gjort det muligt at sparre med andre grupper og TA's under hele projektet. Derimod har Swing ikke tilladt en særlig fleksibel konstruktion af komponenter i brugergrænsefladen; valget af layout, opbygningen og placeringen af komponenter har vakt udfordringer. Fordelen har til gengæld været, at programmet er nemt at udvide, så snart fundamentet for brugergrænsefladen er lagt.

Kapitel 3

Analyse og design af brugergrænseflade

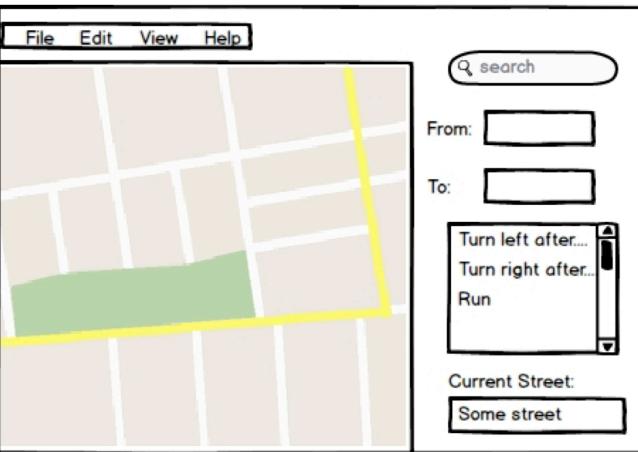
Dette kapitel fokuserer på at gennemgå og diskutere løsningsideer til brugergrænsefladen.

3.1 Designfasen

I designfasen af programmets brugergrænseflade blev der taget højde for en række koncepter introduceret i kurset 'Systematisk opbygning af brugergrænseflader'. Første ide til håndtering af dårligt input fra brugeren var at lave et pop up vindue med en fejlmeddeelse. Dog blev denne håndtering af dårligt input hurtigt genkendt som værende irriterende for brugeren, da man var nødt til at trykke vinduet væk uden at det egentlig gav noget funktionalitet for brugeren. I stedet bliver fejlmeddelelsen vist til brugeren i JListen med instruktioner til rutevejledning. I designfasen af GUI'en herskede der tvivl om, hvorvidt programmet skulle hente inspiration fra Google Maps, OpenStreet Map osv. Vi var tillige i tvivl, om placeringen af søgefelter og knapper skulle ligge ovenpå kortet eller i et separat panel med komponenter. Ud fra 'the Gestalt laws', der blev introduceret i kurset 'Systematisk opbygning af brugergrænseflader', blev panelet valgt til at holde de komponenter, som programmet skulle have. Panelet gør brug af lovene om 'enclosure' og 'continuity'. Ud fra loven om 'enclosure' virker komponenter indenfor samme beholder mere sammenhængende. Som resultat er det mere naturligt for brugeren at opfatte komponenterne som en sammenhængende enhed, der relaterer sig til de samme funktionaliteter (her navigation og rutesøgning). Ud fra loven om 'continuity' opfatter brugeren naturligt ting som sammenhængende, når de er i samme linje men opdelt af mellemrum. Dette gør sig gældende i sidepanelet, hvor labels og komponenter er på samme linje adskilt af mellemrum. Eksempelvis har hver checkbox en label [User Interface Design].

3.2 Virtuelle vinduer

Vi har løbende udviklet en række virtuelle vinduer med mock up værktøjet på mybalsamiq.com. I eksemplet forneden består brugergrænsefladen kun af ét overordnet vindue, hvorfra brugeren kan tilgå de mest gængse funktionaliteter uden at skulle skifte vindue. Funktionaliteterne er klart afgrænsede, så brugeren har en intuitiv fornemmelse af programmets struktur.



Figur 3.1: Mock-up af GUI

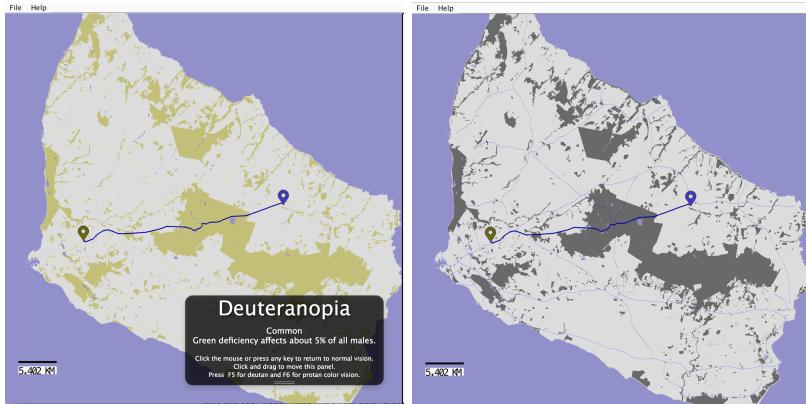
3.3 Endeligt designvalg

Vi har besluttet os for, at programmet kun skal bestå af et vindue med alt information centreret i et sidepanel og en menubjælke. I menubjælken kan man vælge hvilket kort, der skal vises, og om man ønsker at læse sin egen kortfil. I den højre side af programmet, vil det være muligt at søge på enkelte veje og finde dem på kortet eller søge på en rute mellem to punkter på kortet med en ”til-” og ”fra-funktion”. Nedenunder vil der være et tekstfelt, der beskriver vejen fra den angivne startposition til den angivne slutposition gennem en sekvens af instruktioner til brugeren. Det er desuden blevet besluttet, at tastaturet ikke skal have nogen tilknytning til selve kortet. Dette valg er blevet truffet med inspiration fra Google Maps. Herudover har vi valgt, at der på selve kortet skal zoomes ind og ud fra musepilens lokation. Vejnavnene bliver vist ud fra den nærmeste vej til musepilen, når brugeren bevæger musen henover kortet. For at bevæge sig rundt på kortet, skal brugeren benytte musen til at trække sig rundt på kortet. Når en bruger søger efter en vej, vil et valg af relevante muligheder vise sig under tekstboksen, som brugeren skriver i. Det er både til hjælp for brugeren og for programmet. Stave fejl vil på denne måde kunne begrænses, og brugeren kan lettere finde den ønskede vej (auto completion).

3.4 Farveblindhed

Folk med farveblindhed kan ikke skelne mellem bestemte farver, hvilket kan forårsage problemer ved brug af korttjenester. Her spiller farver en afgørende rolle i at skabe et overblik ved at skelne ting fra hinanden på kortet. Dette imødekommer programmet med sin ”Colorblind Mode”knap i sidepanelet. Farverne på vigtige geografiske elementer bliver ændret, når funktionaliteten er slået til. Eksempelvis bliver hovedveje, motorveje og ruteveje farvet blå istedet for gul, som opfattes overvejende anderledes hos en farveblind. For at få en idé af brugeroplevelsen hos en farveblind, har vi benyttet os af tredjepartsprogrammet ”Color Oracle”. Color Oracle er et gratis program, der dynamisk simulerer farveblindhed på ens computer. Dette er blevet kørt på vores kortlægningsprogram for at se, hvornår farverne volder problemer. Vi har vurderet, at det er særlig vigtigt for brugeren ved rutevejledning.

Derfor er farverne skiftet på hovedveje og motorveje som er gule, da gul er tilnærmelsesvis usynlig under kørsel af "Color Oracle". Samtidig er farven for en rute fra A til B blevet gjort tykkere end alle linjer og farvelagt med mørkeblå, så den skiller sig ud fra resten. Dette er gjort eftersom det er en af kerneopgaverne, som programmet skal løse for brugeren uden forvirring [Loung].



Figur 3.2: Illustration af farveblindhed

3.5 Kortvisning

Til kortlægning af Danmarks kort henter vi data fra OSM-filer. Linjerne (ways) som optegner de forskellige ting på kortet er skab til brug på en globus. Programmet skal dog vise kortet i to dimensioner og tage højde for Danmarks breddegrader ligesom på Krak eller Google Maps. Ved at bruge affine transformationer forhindrer vi, at kortdata bliver misvisende og opretholder de samme størrelsesforhold efter den nye skalering i programvisningen af kortet. Processen hvor en kugle bliver projekteret ned på en tangent- eller sekantplan er også benævnt som "ortografisk projektion" indenfor kartografi. Vi har benyttet os af Mercator projektionen, der at stilistiske årsager er valgt til at skalere kortet og gøre det fladt.

3.6 Opsummering

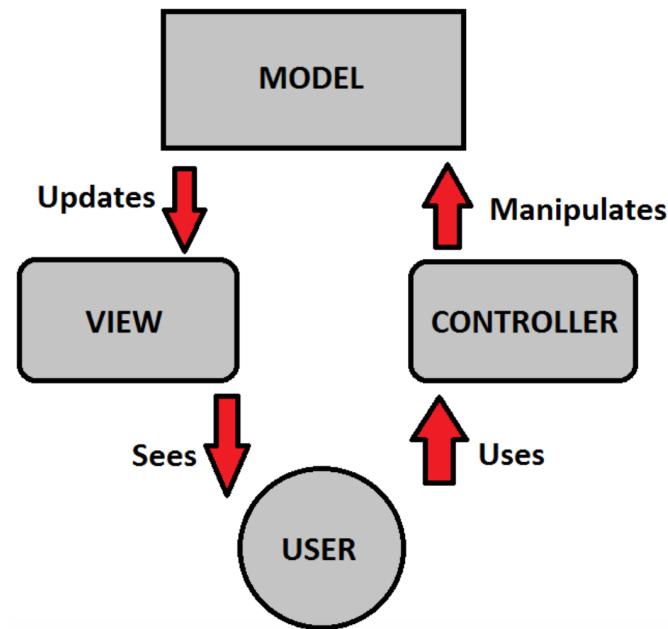
Vi har gennem inspiration fra kurset i "Systematisk design af brugergrænseflader" designet en brugergrænseflade, hvor alt data tilgås fra ét vindue med adgang til de vigtigste funktionaliteter i et panel og en menubjælke. Kortvisningen er skaleret efter Mercator projektionen, og farverne kan ændres i tilfælde af farveblindhed.

Kapitel 4

Teknisk Analyse

4.1 MVC designmønster

Vores program er struktureret ud fra et MVC designmønster. I dette mønster deles håndtering af data, visualisering af data og kontrol af programmets flow op i tre dele; model(data), view(visualisering) og controller(kontrol). Modellen er som udgangspunkt en isoleret enhed, der ikke kender til hverken controlleren eller viewet. Viewet kender derimod til model, da den skal visualisere dataen, som model har. Controlleren kender til både viewet og modellen. Den kontrollerer hvornår og hvad der sker med datastrømmen fra modellen til viewet. En passende analogi svarer til en styrepind i et fly. Dette mønster skaber en struktur med et klart overblik og er med til at programmet får en høj sammenhæng.



Figur 4.1: Visualisering af MVC modellen

4.2 Starter-klassen

Starter-klassens formål er at instantiere programmet med data, brugergrænseflade og brugerinteraktion. Klassen bruges når programmet bliver aktiveret af jar-filen, eller når programmet selv ønsker at skabe et nyt vindue af programmet, hvilket sker når *Bornholm*-knappen bliver kaldt under *File*-menuen. Begrundelsen for at separere dette fra Main-klassen er, at Main-klassen kun bør blive brugt ved kørsel af programmet, og separation gør, at starter-klassen kan kaldes repetitivt. Klassen opererer efter MVC-design strukturen, hvor den først skaber alt data (Model), herefter udseendet (View), og til sidst brugerinteraktionen (Controller). Som undtagelse har vi dog begået en designmæssig fejl, da en *FileChooser* bliver skabt efterfølgende. Den håndterer aktionerne, når brugeren trykker på en af ”File”-menuens valgmuligheder. Dette modstrider MVC-strukturen, da Controller-klassen burde være ene ansvarlig for alt brugerinteraktion - som resultat er arbejdsfordelingen spredt unødvendigt. Begrundelsen for dette designvalg er pragmatisk og skyldes udelukkende, at vi først opdagede det i slutningen af arbejdsprocessen. Hertil har vi valgt at opprioritere bugs, der udgjorde en tydelig trussel mod programmet og dets funktionaliteter.

4.3 Implementering af brugergrænseflade

Alle klasserne under pakken ”View” er ansvarlig for, hvordan brugergrænsefladen kommer til at se ud. Hovedklassen i denne pakke er MapView-klassen, som instantierer og binder alle de resterende komponentklasser sammen i en præsentabel GUI. I forhold til MVC-designet tilsvarer den ”View”-klassen. Før forelæsningen om god kodestil givet af TA Sune Andreas Dybro Debel, var der blot en enkelt klasse i vores program til at bygge komponenterne til vores panel, *BuildGuiComponents*. Denne indeholdt metoder som byggede de forskellige komponenter, getter-metoder til disse og sub-metoder, som gjorde komponent byggemetoderne mere kompakte. Klassesemantikken var misvisende idet navnet var et udsagnsord, der afspejede en handling ligesom metodekald. Herudover dækkede klassen flere ansvarsområder. Efter forelæsningen blev klassen delt op i langt en masse mindre klasser, som hver især håndterede bygningen af de enkelte komponenter. Eksempelvis bør forestilles en klasse, der kun opretter labels, en for textfields osv. Derudover er der lavet en klasse *ComponentsBuilder* som initialiserer byggerklasserne og har getters til de initialiserede klasser. Denne ændring har gjort klasserne langt mere overskuelige og givet dem en generelt højere sammenhæng. Udover dette har størrestedelen af komponent (fx knappen og de to søgefelter) selv fået variabelnavne og getters. Dette har øget kodeforbruget til fordel for en øget forståelighed.

4.3.1 MapView klassen

Formål

MapView er den samlende klasse for programmets View mappe, der nedarver JFrame og implementerer Observable. Klassen er GUI’ens hovedkomponent og er selv en udvidet udgave af JFrame-klassen. MapView implementerer Observable for at kunne kalde repaint() metoden på Canvas, når den modtager brugerinput i form af bevægelser på kortet. JFramet indeholder et BorderLayout, der fungerer som beholder til komponenterne. I borderlayoutets CENTER er der et canvas, hvorpå selve kortet bliver tegnet. I borderlayoutets EAST er der et JPanel, som indeholder en række komponenter til interaktion.

Implementering og komponenter

MapView tager en instans af DatacollectorModel, der bliver initialiseret i sin konstruktør gennem main() metoden og får derigennem data til at tegne kortet. I konstruktøren oprettes der instantier af de forskellige 'byggeklasser' fra View-mappen, fx Canvas, Panel og MenuBar. Idet programmet anvender Swing til programmets GUI er der en række klasser til at bygge mindre komponenter i vores GUI. Eksempelvis sørger 'JLabelBuilder' klassen for at oprette instantier og variabelnavne for de labels, som programmet bruger. Alle 'byggerklasser' der laver komponenter (JComponents) til panelet bliver initialiseret i ComponentBuilder-klassens konstruktør. ComponentBuilder bliver initialiseret i Panel-klassen, som placerer alle komponenterne i et GridBagLayout. Gridbaglayoutet giver en høj fleksibilitet og mange muligheder for at strukturere komponenterne, man anvender meget præcist, så de er fordelt ligeligt. Til gengæld er komponenterne sat til en konstant størrelse, idet gitrene bliver påvirket af størrelsen på de komponenter, som den indeholder. Panelet sættes i det omtalte område i Top containerens borderlayout. Topcontainerens størrelse er sat med udgangspunkt i et format, hvor bredden er 1080 pixels og højden udregnes som værende bredden / 12*9. Til slut anvendes et AffineTransform objekt til at oversætte de læste kortdata fra Datacollectormodel. Objektet konverterer koordinater fra længde- og breddegrader til pixelværdier, som tegnes på canvas.

Ufordringer og videreudvikling

Tidligt i udviklingen af programmet, var der blot en enkelt 'bygger-klasse' (ComponentBuilder). Klassen blev dog meget stor og uoverskuelig, idet den byggede alle komponenterne til panelet selv. Der blev taget en fælles beslutning, om at designe en struktur med flere mindre klasser i stedet for en stor (Clean Code). Denne opbygning Viewet har medført en høj sammenhæng for hver klasse. Som resultat er de respektive klasser i View små, overskuelige og centreret omkring ét ansvarsområde. Slutteligt har strukturen gjort det nemt for udefrakommende at videreudvikle eller ændre brugergrænsefladen. Dette skyldes, at det fremstår klart, hvor specifikke komponenter oprettes og behandles.

4.3.2 Canvas klassen

Formål

Canvas-klassen udgør det tegneområde som bruges til at kortlægge det danske landkort og dets geografiske kortelementer. Klassens vigtigste metode er paint() metoden, der benytter sig af Java's Graphics2D API til at tegne kortet.

Paint metoden

Metoden starter ud med at oprette et rektangel der omkranser kortudsnyttet, som brugeren ser på. I programmet kaldes denne rektangel for *ViewBox*, og alt som er indenfor denne rektangel bliver tegnet. Herefter udregnes afstanden fra venstre side af kortudsnyttet til højre side(I programmet kaldt: *trueZoomScale*, hvilket bruges til at sætte tykkelsen på stregerne og specificere afstanden imellem elementer på kortet (*drawGeoDist-metoden*). Afstandsmålingen og målestoksbjælken bliver først skabt til sidst i paint() metoden. Efterfølgende bruger programmet viewboxen til at få et udsnit af datasættet fra hele Danmark. Dette datasæt og dens elementer indsættes i *pathTagsToBeDrawn*-arraylisterne. Hvis det første element ikke skal tegnes endnu, bliver der ikke hentet et udsnit af datasættet.

Eksempelvis er *Mainroads* det yderste element til at blive tegnet i dens liste, så hvis den ikke skal tegnes endnu, så hentes udsnittet ikke. Originalt var det tiltænkt, at elementer skulle indsættes i HashSets. Eftersom disse fylder mere i forhold til hukommelses forbrug pr. element, så brugte vi arraylister i stedet. Fordelen ved et HashSet ville dog være, at ens elementer ikke ville blive gentegnet (pga nøgle-værdi-princip). Selvom der forekom nogle af de samme elementer i datasættet var programmets hastighed dog ikke mærkbar forbedret. Når de udvalgte elementer er blevet hentet, bliver elementer tegnet efter den rækkefølge, der står beskrevet i *Tags*-Arrayet. Denne liste sørger for at holde samlinger af forskellig type data adskilt (fx. bygninger og veje). Programmet kører derved hurtigere, da det ikke er nødvendigt at skifte farve og/-eller stregstørrelse for hvert gennemløb af listen.

Zoomniveau

Når et element skal tegnes, bliver den først vurderet ud fra kortets zoomniveau (*trueZoomScale*). Med andre ord skal elementets geografiske betydning være relevant på netop det zoomniveau, brugeren befinder sig på. Eksempelvis er det irrelevant at vise bygninger, når brugeren er i det yderste zoomniveau. Hvis elementet skal tegnes, sættes stregfarven og stregtykkelsen, og det bliver bestemt om elementet skal udfylde eller tegne om sit Path2D objekt. Herefter gennemgår programmet den pågældende liste som elementet tilhører. Her evaluerer programmet om det udvalgte element er det element, som skal tegnes. Sideløbende tjekkes der om, hvorvidt elementet ligger indenfor det rektangel, som afgrenser det synlige kortudsnit i programmet. Hvis begge betingelser er sande tegnes eller fyldes elementet på tegneområde og ellers ignoreres det. Koden fra Arrayet med Tags og tegneprocessen kan ses i ”Canvas klassen” under kildekoden i bilag.

Ruten

Til sidst tegner programmet ruten, hvis den eksisterer, opretter målestoksbjælken (*drawGeoDist*-metoden) og sætter start og slutpunktet for ruten, hvis den eksisterer. Derudover, så indeholder klassen metoder såsom: *pan*, *zoom*, *getPointOnMap*, og *panMapCoords*, der tillader Controller-klassen at henholdsvis bevæge brugeren rundt på kortet, finde et punkts geografiske koordinat ud fra musen og sende brugeren hen til en bestemt punkt på kortet. Resten af Canvas-klassen består af hjælpemetoder, fx *drawAddressIcon*, *setViewBox* og *makeFont*. Begrundelsen for at separere Canvas-klassen for MapView eller andre klasser skyldes et ønske om at separere ansvarsområder. Canvas-klassen står alene for at tegne Danmarkskortet. Derudover har andre klasser behov for at kunne tilgå klassen. Eksempelvis når det tætteste vejnavn på musen skal findes, hvorved Canvas-klassens ”*getPointOnMap*”-metode bliver kaldt.

4.3.3 AffineTransform

Indenfor Java findes klassen *AffineTransform*, der repræsenterer et 2D affine transform objekt. Den udfører en lineær transformation fra længde- og breddegrader til 2D koordinater, hvor størrelsесorden, ”rethed” og parallelitet for linjerne er bevaret. Dette bliver brugt til at sikre, at kortet i programmet ikke bliver misvisende jævnfør Mercator projktionen.

4.4 OSMHandler klassen

4.4.1 Formål

OSMHandler-klassen er ansvarlig for at læse OSM-filen og oversætte filens informationer til brugbart data, som indsættes i bestemte datastrukturer. Den nedarver fra Default-Handler-klassen, hvilket gør den i stand til at læse OSM-Filen. Klassen behandler hver linje i filen, og hvis linjen indeholder bestemte datatags, så udfører OSMHandler'en bestemte metoder. Herunder er en opremsning af data tags, som programmet benytter sig, og hvad de indeholder af informationer: "bounds"-taget indeholder minimum og maksimum længde- og breddegraden for udsnittet af filen. "node"-taget er et bestemt koordinat i filen. "nd"-er en reference til en node. "way"-er en samling af "nd"-referencer, der beskriver en linje, vej, bygning eller andre formationer af linjer. "tag"-er ligesom et tillægsord, da den giver yderligere information om sin tilknyttede datatype. Eksempelvis indeholder mange nodes tags. Disse tags afgør om noden fx. er en del af en kystlinje, adresse eller noget helt tredje. Slutteligt er tags kendetegnet ved at have en key (k=) og value (v=) tilknyttet. "Osm"-taget markerer start og slutpunktet for filen.

```
<node id="340539236" lat="55.6892236" lon="12.5897993" version="5" timestamp="2015-01-23T18:31:14Z"
changeset="28356049" uid="2184056" user="AWsbot">
  <tag k="addr:city" v="København K"/>
  <tag k="addr:country" v="DK"/>
  <tag k="addr:housenumber" v="2"/>
  <tag k="addr:postcode" v="1271"/>
  <tag k="addr:street" v="Poul Ankers Gade"/>
  <tag k="osak:house_no" v="2"/>
  <tag k="osak:identifier" v="0A3F507ADBB5328E0440003BA298018"/>
  <tag k="osak:municipality_name" v="København"/>
  <tag k="osak:municipality_no" v="101"/>
  <tag k="osak:revision" v="2002-04-08T00:00:00"/>
  <tag k="osak:street_name" v="Poul Ankers Gade"/>
  <tag k="osak:street_no" v="5584"/>
  <tag k="source" v="OSAK (2010)"/>
</node>
```

Figur 4.2: Udsnit af OSM-fil på en "node"-tag.

4.4.2 Virkemåde

OSMHandler'en behandler indledningsvis "bounds"-taget, der bruges til at skabe grænseboksen (bounding box). Grænseboksen bruges som quadtræernes og kd-træets rod-rektangel og afgør, hvilken data der behandles. Herefter behandler programmet alle nodes i filen, hvilket bliver indsat i hashmappen "ImprovedReferenceMap", så programmet senere hen kan få adgang til punkterne uden at skulle genlæse filen. Hvis punkterne indeholder adresseinformationer, så bliver gadenavn, husnummer og postnummer sat ind i programmets adresseoversigt, hvilket er en TST-datastruktur (præfiks søgetrie) med selve adresse punktet indeni sig. Derudover eksisterer der et hashMap, hvorfra man kan tilgå bynavne ud fra postnummeret. Rationalet har været at programmet skulle kunne tilgå alle adresseinformationer ved at tage postnummeret og finde bynavnet. Istedet for at hver punkt skulle indeholde på bynavnet og andre informationer, der gør den unik. Dermed kunne man optimere hukommelsesforbrugeren. Dette blev dog ikke færdig implementeret, så i det endelige program er det kun gadenavn, husnummer og postnummer, der bliver vist (se eksempel 1 i sektionen "OSMHandler" under afsnippetet "kildekode" i bilag). Programmet behandler forskellige typer data ved at nulstille alt relevant data, når "way"-taget findes i linjerne for en given fil. Dette sker eftersom taget tilsvarer starten på en ny samling linjer (se eksempel 2 under bilag).

Fra startelement ”way” til slutelement ”way” vil OSM-filen indeholde ”nd”-tags, hvilket programmet vil indsætte i en arrayliste kaldet ”smallways”, og den vil samtidigt bygge en Path2D op, der holder på start- og slotpunktet for samlingen. Derudover vil filen indeholde et tag, der indeholder hvilken type samling af linjer det er. Programmet vil kun tage benyttelse af det, hvis det enten er en samling af kystlinje streger, eller hvis det er en vej.

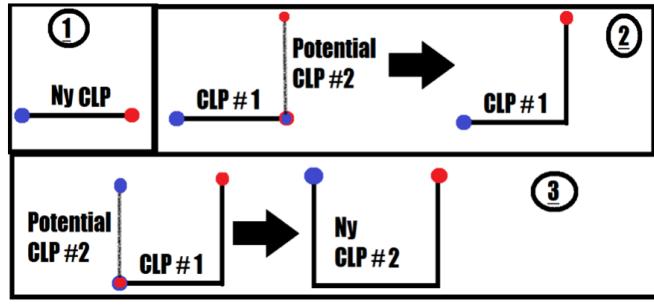
4.4.3 Vejtyper

Programmet starter først med at undersøge om samlingen af elementer er af en bestemt vejtype, når det tilsidst har fundet slutelementet for en way. Hvis dette er tilfældes indsættet det tilknyttede vejnavn i ”addresslist”. Herefter behandler programmet alle punkterne i ”smallways” ved at indsætte dem i Kd-træet med deres tilhørende punkt og en ”integer”-reference. Referencen afgør hvorhenne i ”addresslist”, at punktet kan finde dets vejnavn. Hvis vejen ikke har et navn, så får punkterne 0 som ”integer”-reference, hvilket i listen altid er ”Unavngivet vej”. Fordelen ved denne fremgangsmåde er, at programmet sparar en masse hukommelse. Dette sker ved at gemme en reference til navnets placering, så programmet kun holder på få instanser af navne - stedet for at flere punkter holder på samme navn. Dog tillader den eksisterende struktur, at der kan forekomme duplikationer af det samme vejnavn i ”addresslist”. For at forhindre dette kan man opsætte en ”contains”-metode, der først evaluerer om navnet eksister i listen. Derefter bruge en ”indexOf”-metode til at finde positionen af navnet. Vi har dog vurderet, at køretiden for databehandlingen i OSMHandleren sted med 1-2 minutter, så vi fravalgte det og holdt fast i den nuværende struktur.

4.4.4 Kystlinjer

Hvis samlingen af linjer ikke er veje, men er en samling af kystlinjer vil programmet forsøge at udføre en af følgende tre scenarier:

1. Først vil programmet gennemgå alle kystlinjesamlinger, og hvis ingen af dem starter eller slutter på den nyes samlings start eller slotpunkt, så skabes der et nyt objekt kaldet: ”CoastlinePathTag”. (Forkortet: Det nye objekt indsættes i samlingen af kystlinjesamlinger, hvilket er i en arrayliste kaldet: ”CoastlinePathTags”. ”CLP”), der holder på start og slotpunktet, samt Path2D’en, der udgør samlingen af kystlinjer.
2. Hvis programmet finder en CLP, hvis slotpunkt er starten på den potentielle nye CLP, så ”appender” (tilføjer) CLP’en sin egen Path2D til den potentielle nye CLP’s Path2D og erstatter sit slotpunkt med den potentielle nye CLP’s slotpunkt.
3. Hvis programmet finder en CLP, hvis startpunkt er slutningen på den potentielle nye CLP, så ”appender”(tilføjer) den potentielle nye CLP med sin Path2D med CLP’ens Path2D. Den erstatter herefter sit slotpunkt med CLP’ens slotpunkt og fjerner herefter CLP’en fra listen af ”CoastlinePathTags”. Til sidst indsættes den potentielle nye CLP i listen.



Figur 4.3: De 3 scenarier der kan forekomme, imens kystlinjerne forbides. Den blå cirkel er startpunktet og den røde cirkel er slutpunktet.

4.4.5 Veje

Hvis samlingen af linjer ikke er kystlinjer men veje, så indsættes de passende efter type i programmets quadtræer. Dette sker ved at listen af ”smallways” tager et element fra dens liste og elementet der kommer efter den, og laver en vej ud fra den, som herefter indsættes i quadtræet. Fordelen ved at dele vejene op i små linjesegmenter kommer til udtryk gennem programmets evne til at finde en given korteste eller hurtigste rute. Ulempen er til gengæld at quadtræerne fylder mere, end hvis de holdt på den samlede vej istedet. Ideelt set burde hver vej indsat i quadtræet indeholde et array med småveje (edges), så den korteste og hurtigste rute fortsat kan findes. Til sidst i ”way-taget” så bliver alle de relevante informationer nulstillet. Når programmet når slutningen på ”osm-taget” (slutningen af filen), gennemgår den listen af kystlinjesamlinger igen, og forsøger at binde dem sammen i en større enhed. Herefter indsættes de samlede kystlinjer i quadtræet ”absoluteTree” og listen ”coastlinePathTags”. Hashmappet ”ImprovedReferenceMap” bliver nulstillet, da informationerne ikke længere er brugbare på dette tidspunkt.

4.5 Serializable

En af udfordringerne ved at tegne hele Danmarks kortet opstår ved databehandlingen i OSMHandler-klassen. Her bearbejdes indledningsvis en OSM-fil for hele Danmarks kortet, der holder på 12 millioner linjestykke i et XML-format. For at optimere plads- og tidsforbruget kan man ændre det format filen gemmes i. Eksempelvis fylder Danmarks kortet omkring 10GB som OSM-fil men kun 600 mb i zip format og er ydeligere nedsat til 200 mb i et binært format. Interfacet ’Serializable’ i Java tillader hurtig læsning og skrivning af denne kortdata.

4.5.1 Definition

Serialization dækker over omdannelsen af objektdata til en sekvens af bytes, der skrives til en fil (serialize) eller læses fra en fil (deserialize) i programmet. Et objekt repræsenteres som en sekvens af bytes, der indeholder objektets data samt oplysninger om objektets type og de typer af data, der lagres i objektet (felter). Efter et objekt er blevet serialiseret og gemt i en fil, kan den læses fra filen ved at genskabe objektets ”hukommelse”. For at muliggøre denne proces i Java, skal klassen som producerer en instans af objektet selv implementere Serializable i klassesignaturen.

4.5.2 API

Serialization interfacet holder ikke selv på nogle metoder eller felter, men benyttes udelukkende til at stedfæste, hvilket klasser (og subklasser), der understøtter objekter at blive skrevet og læst som en sekvens af bytes. For at påvirke, hvad der bliver gemt undervejs skal metoderne `writeObject()` og `readObject()` implementeres. Førstnævnte er ansvarlig for at gemme objektets nuværende datatilstand for en given klasse. Sidstnævnte bruges til at genskabe en kopi af den selvsamme data. Objektets tilstand gemmes ved at skrive de enkelte felter til `ObjectOutputStream`, der bruger `writeObject()` metoden.

4.5.3 Kortdata

I vores nuværende implementation af programmet, gemmes kortdata forskelligt afhængig af, hvad det repræsenterer på kortet. Alle punkter som tilsammen udgør linjesegmenter eller forskellige vejtyper er gemt i quadtræer. Adresser bliver gemt i en trie, hvor hver adresse peger på et `Point2D` objekt. Programmet benytter også et KD-træ til at navigere frem til alle vejpunkter. Det rektangel der afgrænser vores tegneområde bliver også gemt. Til slut gemmes en liste af alle gadenavne og liste på alle kystlinjer.

4.5.4 Implementation

For at gemme ovennævnte data har vi valgt at oprette tre klasser, der tilsammen gør det muligt at gemme og læse data dynamisk. `FileSaver`-klassen i `Model`-pakken bruges til at gemme kortdata (se 1.5.1). `FileLoader`-klassen i `Model`-pakken bruges til at læse data fra en binær fil med kortdata (se 1.5.2). `FileChooser`-klassen i `Controller`-pakken tillader brugeren at læse fra en given kortdata fil på sin computer og gør det muligt for brugeren selv at afgøre kortvisningen. Dette gør `FileChooser` ved at lytte på vores `save` og `load` knapper i `Viewet`, som bruger metoderne `saveBinary()` og `loadBinary()`. Se eksempel 1 og 2 for uddrag fra koden for henholdsvis begge metoder i `FileSaver` og `FileLoader` klassen.

4.5.5 Fordeler og ulemper

Det at vi gemmer vores kortdata binært vha. `Serialization` i Java er først og fremmest en fordel, da det tillader programmet at læse og skrive data hurtigt. I forbindelse med implementeringen, har vi dog også mødt en del udfordringer. Som resultat af en formentlig fejl i Javas API, har vi været nødsaget til at oprette en ”stråmandsklasse”, der kunne fungere som mellemmand ved lagring af `Path2D` objekter. Helt specifikt oplevede vi problemer med at serialisere `Path2D` objekter, hvilket sandsynligvis skyldes dens private konstruktør [Overflow]. Ifølge indlægget på StackOverflow skyldes det, at Java `Serialization` ikke kan tilgå konstruktøren, så enhver klasse der nedarver fra `Path2D` ikke understøtter `Serialization`. For at imødekomme problemet har vi lavet mellemmanden ”`PathTag`”, der bruger `Path2D.Float` som felt istedet for at nedarve fra klassen. Derudover er det bemærkelsesværdigt, at den gemte data ikke kan læses, hvis der er foretaget ændringer i de klasser, der skrives fra. Slutteligt kan man ikke tilgå konstruktøren i et objekt, som er blevet genskabt (deserialized). Interfacet `Serializable` mindske altså fleksibiliteten i programmet, idet man ændrer klassens interne struktur. Dette har vi dog vurderet som et mindre trade off sammenlignet med de voldsomme pladsbesparelser.

4.6 Datastrukturer

Indenfor softwareudvikling er datastrukturer en måde at organisere ens data på i et program, så den kan bruges effektivt. Altafhængig af programmet, arbejdsopgaver og formål vil valget af datastrukturer imødekomme forskellige behov. I grundlæggende programmering har vi tidligere været vant til at benytte os af lister med en lineær opbygning til at opbevare vores data (fx. Array). Dette kan dog være en ulempe, hvis man skal søge igennem store mængder data, hvorved det vil tage lineær tid ud fra størrelsen på ens liste. For at optimere søgetiden kan man med fordel overveje andre datastrukturer. Eksempelvis er egenskaberne ved datastrukturen for et træ anderledes, da det er hierarkisk opbygget og tillader effektive søgeopslag. I følgende emne behandles datastrukturen for et Quadtræ med en beskrivelse og forklaring af dens virkemåde, metoder og fordele og ulemper.

4.6.1 Quadtræ til kortdata

Formål

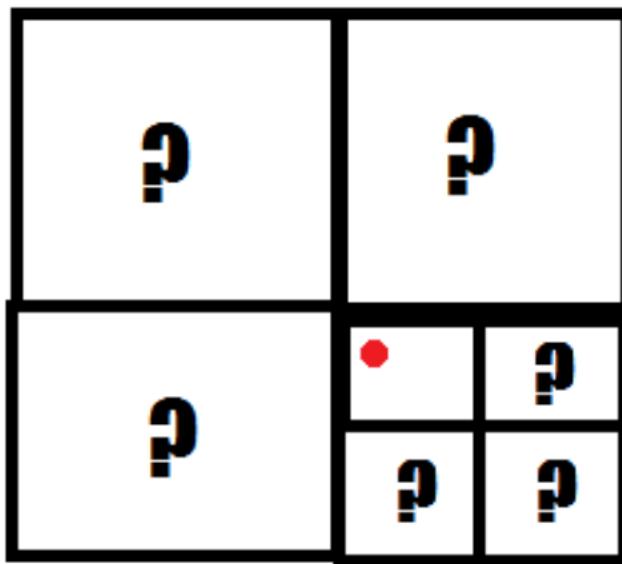
En af hovedudfordringerne ved kortlægning af Danmark er performance-relateret. Uden effektive datastrukturer og filtrering af hvad der tegnes, skal programmet loade omtrent 12 millioner linjer hver gang danmarkskortet optegnes. For at opleve en gnidningsfri overgang mellem opdateringerne af kortet, bliver man derfor nødt til at bruge effektive datastrukturer og filtrere ting fra på kortet afhængig af zoomniveauet. Som datastruktur kan man bruge et quadtræ, hvis knudepunkter enten er blade eller har fire børn. Quadtræet baserer sig på del-og-hersk princippet, som er kendtegnet ved at et overordnet problem løses ved at opdele det i mindre subproblemer, der er trivielle at løse.

Virkemåde

Quadtræet er ansvarlig for at holde på alle de elementer, som skal tegnes på kortet. Træet indsætter løbende alt information i nye knudepunkter repræsenteret af rektangler. Som udgangspunkt er der et overordnet rektangel med alt data, der rekursivt bliver firdelt til ydeligere rektangler med hver sin kompasretning (se figuren på næste side). Klassen får hjælp fra en Node-klasse, der gør det muligt for quadtræet at agere som en træstruktur. Hver gang et rektangel (et knudepunkt, fx rod'en) bliver firdelt, skabes der fire nye subtræer. Hver subtræ er repræsenteret som et knudepunkt, der holder på en liste af elementer indenfor sit respektive rektangel. Hvert rektangel er enten instantieret eller sat til null, hvis de endnu ikke er firdelt. Når træet oprettes, modtager den en grænseboks som rektangel for rod'en (bounding box) sammen med et tal på, hvor mange antal elementer, der maksimalt er tilladt i hvert knudepunkt. Hvis en grænse ikke er fastsat for opdelingsniveauerne, vil firdelingen forekomme i uendelighed. Når programmet skal modtage informationer fra træet, søger træet efter data altafhængig af kortudsnyttet og det rektangler. Træet finder det rektangel, som omkredser kortudsnyttet og deler derfor kun den data, der er relevant under kørsel af programmet.

Grænser

Træets grænser tilsvarer den bounding box, som omkredser kortet i programmet. Det vil sige, at grænserne som udgangspunkt er fastsat til at indramme hele billedet inden opdeling. De fire subtræer som svarer til den første underopdeling af hele billedet er defineret som 4 noder; NW, NE, SE, SW - her repræsenteret med geografiske retninger.



Figur 4.4: Illustration af quadtræets opdeling af kortet

Insertion

Insertion-metoden modtager et PathTag-objekt, hvor både objektet og dens Path2D ikke må være null. Derudover skal dens Path2D være indenfor rodrektalet af træet. Derefter kaldes insert-metoden, som kun quadtræet selv kan accesse. Denne metode modtager et PathTag-objekt og den node, som elementet skal indsættes i. Programmet starter altid med at indsætte elementet i rodren af træet. Hvis rodren / noden er quadret, så lokaliserer programmet hvilken retangle / node, den er inde i. Herefter gentager processen sig indtil, at noden / rektanglen ikke er quadret. Derefter indsættes elementet i noden / rektanglen, og hvis antallet af elementer er oppe på træets maksimum, så quaderer programmet noden, og kalder insert-metoden på noden med alle dens elementer. Derefter fjernes alle elementerne i noden, så det kun er dens subtræer / subnoder, der har elementerne i sig.

Query2D

Query2D-metoden modtager en Rectangle2D.Float fra programmet, som ikke være lig null. Herefter skaber programmet den liste, som skal blive fyldt op med elemmenter, og træet kalder derefter Query2D-metoden, som kun quadTræet kan accesse. Metoden modtager en Rectangle2D.Float, noden den skal query, og listen af elementer, der til sidst skal retunereres. Når metoden skal query, så fungerer den ligesom insertion, da den først tjekker om noden / rektanglen er quadret. Hvis ja, så undersøger den om rektanglen overkrydser nogle af subtræernes rektangler / noder, og hvis den gør det, så kaldes Query2D-metoden på den pågældene rektangel / node. Hvilket programmet vil gentage recursivt for alle noderne den overkrydser. Når programmet finder en node / rektangel, som ikke er quadret, så indsætter den alle elementerne i noden overi listen, der ultimativt bliver returneret til brugeren, og når alle noderne rektanglen overkrydser er nået til dette punkt, så returnes listen.

Implementation

Implementationen af et quadtræ er taget fra bogen 'Algorithms 4th edition' af Robert Sedgewick, som vi herefter har modificeret til at kunne modtage streger og ikke kun punkter [univericty]. [Sedgewick]

Køretid

Det værste tilfælde for søgningen i et quadtræ kan illustreres med følgende scenarie. Man kunne forestille sig at punktet man søger efter befinder sig i det nederste hjørne af en given kvadrant i træet. I dette tilfælde vil man ved søgning være nødsaget til at underopdæle kvadranten rekursivt rigtig mange gange. I så fald vil man opleve at et søgeopslag i datastrukturen, der kan tage helt op til $O(N)$ istedet for $O(\log N)$, hvor N er antallet af elementer. Det værste tilfælde er heldigvis ikke fremtrædende i vores tilfælde, da data for danmarkskortet er statisk. Vi har på grund af dette valgt at implementere quadtræet, som er nemmere at implementere sammenlignet med andre træstrukturer. Værd at bemærke er også, at vi generelt har efterstræbt at afprøve datastrukturen på konkret data istedet for at tage udgangspunkt i det værste tilfælde.

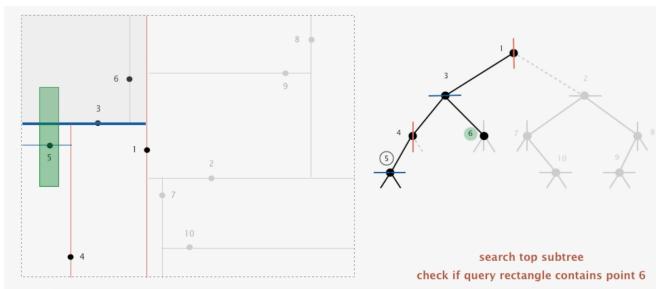
Fordele og ulemper

Quadtræer gør det muligt at gennemløbe og søge data effektivt ved at finde vores grafiske data rekursivt. Hvis vi er interesseret i at finde en tilfældig vej i København, ville vi normalt være nødsaget til at sammenligne denne vej med alle veje i kortet (lineær tid). Fordelen ved quadtræet er, at vi kan afgøre hvilken kvadrant, vejen befinner sig i og rekursivt opdæle denne kvadrant ydeligere. Når datamængden er på et tilpas niveau (fx. 1000 punkter) kan vi søge efter vejen eller punktet uden at skulle gennemløbe alt omkringliggende data. Datastrukturen gør det altså muligt at afgrænse det data, som vi er interesseret i og tegne kortdata effektivt. En af ulemperne ved datastrukturens præstation er dog, at den afhænger af om dets data ændrer sig dynamisk. Med andre ord kan det blive et problem, hvis dataet ændrer sig ofte, fordi quadtræet i så fald vil foretage uhensigtsmæssige opdelinger. Hvis datasættet flyttes rundt, skal træet opbygges igen. Eksempelvis vil et træ med 1000 datapunkter i en knude skulle ændres, hvis der pludselig tilføjes et ydeligere element og grænsen er sat til 1000. Omvendt vil datastrukturen have svært ved at håndtere, hvis den har lavet en masse opdelinger ét sted og dataen derefter bliver rykket. Begrundelsen for ikke at implementere et Kd-træ var pragmatisk funderet, idet quadtræet var nemmere at implementere. Samtidig var det nemmere at modificere, således at det kunne holde på linjer (Path2D) frem for punkter, og den kunne modtage større mængder af data gennem en knude (subtræ med rektangel). Dette skyldes at Quadtræet kun skal forholde sig til mængden af data og opdeler sig selv statisk ud fra rektanglet, der omkredser datasættet. Kd-træet er mere dynamisk ved konstruktion af data, da den i højere grad skal forholde sig til dataplacering og udformning.

4.6.2 KD-træ til adresser

Definition

Et kd-træ er et binær søgetræ, der gemmer punkter i et k-dimensionelt rum. I vores program bruger vi et 2-dimensionelt træ til vores addresser. Når programmet indsætter et nyt element, fungerer den ligesom en BST (Binary Search Tree), med den undtagelse at programmet skiftevis kun vurderer det ene element i et givent punkt (X eller Y). Det ene koordinat bruges til at indsætte det nye element til højre eller venstre i træets punkter. Samtidigt med dette skaber træet rektangler/RectHV [univericty], der indeholder det nye punkt, samt alle eventuelle underpunkter. Dette er særligt nyttigt, når programmet senere skal lokalisere det nærmeste punkt i træet for et tilfældigt punkt vha metoden *nearest*.



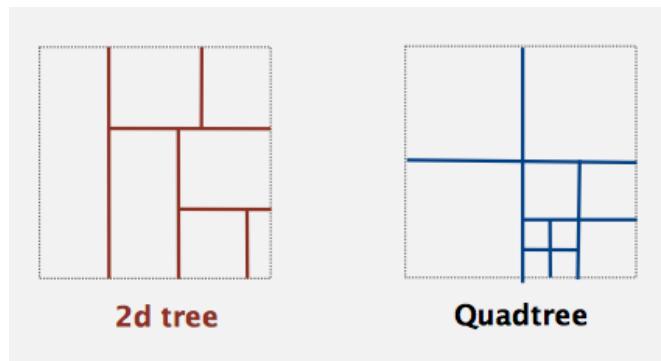
Figur 4.5: Illustration af k-d træets datastruktur

Nærmeste punkt

Når *nearest*-metoden bliver kaldt, starter programmet med at tjekke for om Kd-træets rod-punkt er tættere på end det tidligere punkt. Indledningsvis er der ikke andre kandidater, så roden bliver sat til at være det tætteste punkt. Herefter tjekker træet om rodpunktets rektangel har en mindre afstand (distancetoSq) til punktet. Hvis dette er tilfældet, tjekker træet om det søgte punkts X eller Y-værdi er mindre end værdierne i det nuværende punkt (her roden). Er dette tilfældet, så går programmet ned igennem venstre del af træet først. Hvis det søgte punkts X eller Y-værdi er større end det nuværende punkt, går den ned af højre side. Programmet gentager processen, som beskrevet overover, og tjekker om det punkt den har fat i, har en mindre afstand til det søgte punkt. Hvis ja, så er det tætteste punkt. Processen gentager sig cirka \log^*n gange, eftersom træet skal foretage en todelt søgning for alle rektangler med søgepunktet indeni. Når programmet finder rektangelet, laver den sine sædvanlige checks og begynder derefter at kalde sig rekursivt. Undervejs kalder træet ”nearest” på venstre side, når den går til højre i træet og omvendt. Dette øger køretiden med det dobbelte, da den som minimum skal undersøge begge sider af alle subtræer, der bliver gennemløbet. Når programmet kalder *nearest()* på tilbagevejen, vil den ikke nå langt ned i subtræerne, da den korteste afstand umiddelbart er fundet. Subtræernes afstand til søgningspunkt er derfor større, hvilket får programmet til at returnere det korteste punkt. Den amortiserede køretid for denne operation er: $O(\log(n))$, hvor n er antallet af punkter i Kd-træet. Det værste tilfælde for operationen er $O(n)$, hvilket opstår afhængig af søgepunktets placering i sit respektive rektangel. Hvis det forespurgt punkt ligger langt væk fra punktsættet i rektanglet, skal alle elementerne med samme radius potentiel gennemløbes. I så fald kan det tænkes, at alle elementer skal gennemløbes.

Valg af datastruktur

Begrundelsen for valget af et Kd-træ istedet for et QuadTree skyldes nearest() metode, der er nemmere at implementere. For at få ”nearest” til at fungerer med quadtræer, så kunne man f.eks. lave en metode, der regressivt gik igennem QuadTree, indtil man finder den mindste boks med søgningspunktet indeni, hvilket vil tage $\log(n)$ -søgninger. Herefter sammenligner man med punkterne deri, hvilket man kunne gøre ved at lave en mindre firkant eller cirkel omkring punktet, og derefter tjekkede indeni den for hvilken af punkterne, som har den mindste afstand til søgningspunktet og returner det. Problemet med sådan en fremgangsmåde er, at hvis søgningspunktet befinner sig i en tom quad/kasse, så er der ingen punkter at sammenligne. Hvis man skulle finde et punkt, så ville man være nødsaget til, at gå ud i andre quad’s for at finde det nærmeste punkt. Hvilket rejser en del spørgsmål, for hvor mange quads skal man gå tilbage, og hvis man blot går en quad tilbage, er det nærmeste punkt så garanteret indeni en af dem? Dette er ikke et problem for Kd-træet, der igennem den sin ”nearest” metode, altid har en kandidat klar, der med garanti altid er den, som er tættest på.



Figur 4.6: Quadtræ firdeler på et rektangel mens kd træ todeler ud fra data henholdsvis lodret og vandret

Kildekodehenvisning

Vores version af Kd-træet er udviklet af [Chen]. Dette skyldes primært, at der ikke var meget tid til at udvikle vores egen udgave af Kd-træet, så vi genbrugte hans version og modificerede den til vores behov. [\[learnItKd\]](#)

4.6.3 Dijkstra til rutesøgning

Formål

Programmets rutefunktion finder en vej fra en given adresse til en anden ud fra et givent brugerinput. Funktionen vælger som udgangspunkt den korteste vej fra A til B i bil. Den 'hurtigste' vej vælges tager som forskel højde for fartgrænsen. Derudover kan motorveje undgås. Små parkveje og lignende er automatisk ikke medtaget i rutevejledningen, hvis man er i bil og på samme måde er motorveje automatisk undgået, hvis brugeren er på cykel eller gåben.

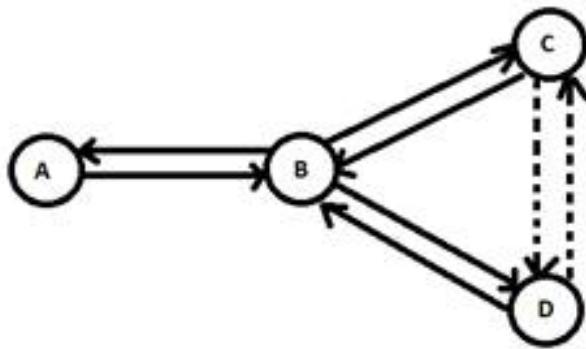
Implementation

Programmets rutefunktion og rutevejledning bruger en modificeret form af Princeton's Dijkstra's algoritme, som anvender en minimum prioritets kø. Modifikationerne kommer til udtryk ved, at algoritmen stopper med at afspænde (relaxe) kanter, når den når destinationsknuden (fra klassens konstruktør). Dette er blot gjort for at undgå, at Dijkstra algoritmen kigger på hele grafen, i tilfælde af den korteste vej til destinationsknuden bliver fundet tidligt i processen. Derudover er "DirectedEdge"-klassen om-skrevet en smule til at kunne holde på en instans af et path2D objekt. Dette er gjort med henblik på nemt at kunne 'oversætte' den fundne sti af orienterede kanter til path2D objekter, som programmet kan behandle på kortet. Derudover holder en orienteret kant (oversat directed edge) også på en int, der angiver rutens retning. Hvis den er 0, går den fra startpunkt til endepunkt på et givent pathTag objekt omvendt, hvis værdien er 1. Dette er gjort for at gøre rutevejledningen mulig, da det er nødvendigt at vide hvilket gradtal, der forekommer fra og til startpunktet. Det varierer, alt efter hvilken vej path2D objektet holder på.

ShortestPath-klassen

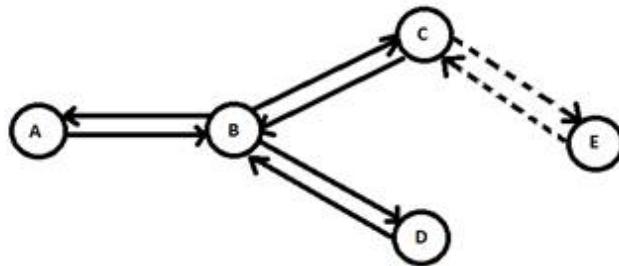
Klassen 'ShortestPath' tager en instans af Datacollection, hvorfra den tilgår data tilknyttet vejene på kortet. Metoden 'findShortestPath()' tager to point2D.Float objekter; punkterne for adresserne, som der skal findes en rute imellem. I metoden hentes path2D objekter for vejene ud i tre forskellige arrayLister; en for stier og små veje, en for hovedveje og en for motorveje. I tilfælde af grafen ikke er bygget på forhånd, sker dette i metoden 'buildVerticesAllRoads', der tager de tre omtalte arraylister som parametre. Alt efter hvilke indstillinger brugeren har givet, vil en boolsk evaluering påvirke hvilke af vejene grafen bliver bygget med. Der er tre forskellige måder hvorpå kanterne for grafen oprettes, men fire forskellige tilfælde. Ved konstruktion af grafen løbes alle pathtags igennem, og der foretages et tjek i hashmappet 'PointToVertex' fra Point2D til int. Her repræsenterer point2D objektet et start- eller endepunkt i et path2D objekt, og int variablen repræsenterer en knude i grafen. Alt efter om begge, ingen eller ét af punkterne optræder i grafen allerede, oprettes der nye kanter og knuder på en måde, der tilsvarer rutens kontekst. Hver gang der oprettes nye kanter og knuder, tilføjes de til det førstnævnte HashMap og en ArrayListe for både knuder og kanter. Se figurerne på næste side for illustration af, hvordan knuder og kanter bygges:

I figur 4.7 optræder begge knuder allerede i grafen, når vi prøver at tilføje dem som kant. I så fald oprettes der en kant mellem de to knuder hver vej, og der oprettes ingen nye knuder:



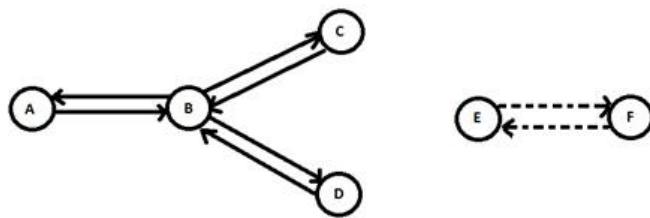
Figur 4.7: Kanter oprettes mellem eksisterende knuder

I figur 4.8 er en af knudepunkterne, som ønskes tilføjet allerede kendt, imens den anden forekommer ukendt (retningen for punkterne i path2D objektet er underordnet). I så fald tilføjes der en ny knude til grafen, og der oprettes en kant mellem den eksisterende og nye knude:



Figur 4.8: Ny knude og kant til eksisterende knude

I figur 4.9 optræder ingen af knudepunkterne i grafen. Derved oprettes de nye knuder med passende indekstal, og der oprettes en kant hver vej:



Figur 4.9: Knuder oprettes og forbindes med ny kant

Vægtet graf

Efter all pathtags er løbet igennem og knuder er bygget, oprettes der en instans af 'EdgeWeightedDigraph' klassen med samme størrelse som antallet af knuder. Herefter tilføjes alle kanter fra arraylisten for kanter til grafen. Dernæst findes de vejpunkter, der ligger tættest på de givne adressepunkter ved 'brute force'. Dette sker ved, at alle knuder i grafen gennemløbes for at finde det givne vejpunkt, der foreligger tættest på henholdsvis hver af de to givne adressepunkter. Dette er en nødvendighed, da adresse- og vejpunkter ikke tilgås i samme datastruktur (de ligger adskilt). Når grafen er konstrueret og den passende start- og slutknude er fundet, oprettes der en instans af 'DijkstraSP'-klassen. Denne instans bruges til at finde den korteste vej vha. grafen, startknuden, slutknuden og en arrayliste af orienterede kanter (directed edges) med en tilhørende vægt.

Udfordringer

Under implementeringen af rutesøgning og rutevejledning blev Princeton's implementering af algoritmen anvendt først, hvilket medførte nogle udfordringer. Problemerne bestod i 'oversættelsen' af kanter til punkter og stier på kortet, som reelt kunne anvendes. For at imødekomme dette har vi måtte om-skrive algoritmen, som ikke anvendte en egentlig graf men opbyggede den med sine kanter, knuder og "backtracking". Dette resulterede i nogle uhåndgribelige fejl, og det blev anbefalet af en TA at anvende en egentlig graf, som er mere overskuelig at debugge og ændre. Derfor blev Princeton's implementering af Dijkstra's algoritme anvendt på ny med den eneste forskel i form af en vægtet graf. Grafen blev brugt til at bygge bro mellem orienterede kanter for en given korteste rute og de værdier, som hang sammen med kortet. Når der ændres transportform eller brugerne gerne vil finde den hurtigste vej frem for den korteste vej bygges hele grafen endnu en gang, hvilket påvirker køretiden. Man kunne bygge og holde forskellige grafer for de forskellige transportformer, hurtigste/korteste vej osv. Dette ville optimere rutesøgningen, så snart alle graferne er bygget med et trade off på pladsen. Da programmet allerede var utsat for store ressourcekrav blev den sidste løsning travlagt.

Køretid

Den totale asymptotiske køretid for Princeton's implementering af Dijkstraalgoritmen er uden opbygning af graf $(E + V \log(V))$. Denne implementering sørger for at knuder som allerede er besøgt kun besøges, hvis kanten til dem en lavere distTo værdi. Til sammenligning er køretiden for den originale Dijkstras algoritme $O(V^2)$, i tilfælde af at hver knude er forbundet med sig selv og alle andre knuder. Dette forekommer dog aldrig med data for Danmarks kortet. Den totale asymptotiske køretid for at finde en rute i programmet uden opbygning af graf er $(E + V + V \log(V))$, idet knuderne skal gennemløbes en ekstra gang for at finde de mest nærliggende til adressepunktet. Med opbygning af graf skal antallet af pathTags, antallet af knuder og antallet af kanter gennemløbes med en samlet køretid på $(p + e + v + (E + V \log(V)))$

4.6.4 Rutevejledning

Rutevejledningen går ud fra et spektre af grader fra den tidligere retning til den man drejer ind på. Spektret er sat ud fra 360 grader som ses på figur y. Alt efter vinklen mellem en path2D's to punkter, gives den en retning, og når man møder en ny vej tjekkes den tidligere retning for at kunne bestemme hvad den nye retning skal kaldes (højre, venstre, ned eller op). Er den tidligere retning fx 'venstre' og den nye retning 'op' vil rutevejledningen bede dig om at køre til højre. Der er en klar fejlmærk i nogle tilfælde, fx hvis en vinkel er lige på grænsen mellem to retninger bliver den givet den en retning konsekvent. Derudover er der eksempler hvor vejen går mod højre og derefter drejer til venstre men ikke når ud af højre spektret og derfor beder brugeren om blot at fortsætte. Programmet tager også kun højde for sving hvis vejnavnet skifter, hvilket kan give problemer hvis et vejnavn skifter umiddelbart før et skarpt sving, da svinget da vil være på samme vej som allerede er blevet givet en vejledning og svinget er da blot med som 'kør ad eksempelvez i 200m'.

4.7 Addressesøgning

I løbet af projektet er addressesøgningen håndteret på to forskellige måde; at håndtere addressesøgning med regulære udtryk eller automatisk udfyldning. Den første implementation af addressesøgning anvendte regulære udtryk, hvilket giver brugeren friheden til, at kunne søge på hvad som helst. Denne implementation medfører dog problemstillingen for dårligt input fra brugeren, hvilket er en større opgave i sig selv at håndtere. En række metoder skulle afvise, rette og tjekke de givne input fra brugeren for at forhindre dårligt input i at generer programmet. Mængden af dårligt input der skal tages højde for er dog enorm, derfor anvendes et eksternt bibliotek i den anden implementering. De regulærer udtryk anvendes som en hjælp til programmet til at finde den korrekte adresse, mens den automatiske udfyldning skal hjælpe brugeren.

Første udfordring ved implementationen med det eksterne bibliotek er hvorvidt den fungerer bedst med en JComboBox eller et JTextField. Ved anvendelse af en JComboBox viser programmet den auto udfyldning, der gav brugeren gode muligheder og i og for sig udelukkede dårligt input i det brugeren skulle 'vælge' et element fra JComboBoxen. Af denne årsag anvendes den anden implementation med et eksternt bibliotek indtil et sent stadiet i udviklingen af programmet.

4.7.1 Glazed List

Det første bibliotek der bliver testet med hedder Glazed List. Dette er en udmærket erstatning, da mængden af addresser der skal gives til vores JComboBox er relativ lille. Til gengæld er JComboBoxen ikke så pæn, som et JTextField. Derudover skal Glazed List køre på en anden tråd for at fungere. Når programmet bliver sat til at køre store filer, er mængden af data for stor til, at dette bibliotek kan bruges, da det forsinket opstarten af programmet. Derfor er denne løsning blevet fravalgt.

4.7.2 Trie

Vores Glazed List fungerede dårligt for et stort input. En Tenery Search Trie implementeres istedet, hvilket tillader programmet at søge på ord med et specifikt præfiks og understøtter en række specifikke metoder til adressesøgning. Et tenery search trie anvender princippet bag et binært søgeræt med en almindelig trie. Den valgte søgertrie er valgt, idet den er mere pladseffektiv end en normal søgertrie. Dog tager det længere tid at lave søgeropslag; $O(L \log k)$ hvor L er længden af ordet. I en normal søgertrie tager det $O(L)$, hvor L er længden af ordet. Her er hukommelsesbrug prioriteret over tid for at finde ordet, idet mængden af adresser er meget stor.

4.7.3 SwingX

Med en optimeret datastruktur til at holde adresserne (Tenery Search Trie) skal auto udfyldningen sammenkobles med brugergrænsefladen. Autoudfyldningen implementeres i et JTextField frem for en JComboBox. På baggrund af eksterne kilder anvendes biblioteket SwingX, hvilket indeholder en videreudvikling af Java's eget. Vha SwingX anvendes et JTextField, der opdateres med et givent input. SwingX bruger ligesom Glazed List en liste, hvilken bliver givet fra programmets søgertrie. Når brugeren indtaster et bogstav gives en adresse fra trien, og adresserne med præfiks til det indtastede tilføjes til SwingX listen. Bemærk at det først er ved indtastning af to bogstaver eller over, at dette sker. Dette gør, at SwingX maksimalt skal holde det antal adresser i sig, som der har præfikset skrevet af brugeren. Dette er tilsvarer i værste tilfælde alle adresserne på kortet, men er usandsynligt givet 'normal' kortdata.

4.7.4 Udfordringer og særlige valg af opsætning

Den endelige implementering har stadig en række problemer, fx benytter JTextField sig af en documentListener, da den ikke selv kan registrere ændringer. Dette påvirker autoudfyldningen, som SwingX giver brugeren, der bliver registreret af documentListeneren. Eksempelvis kan brugeren indtaste "ar" som documentListener registrerer, hvorved der hentes et resultat ud fra søgertrien. Resultatet er "Arendalsfade 2 København 2790", hvilket vises i det benyttede JTextField, og feltet reagerer på det givne resultat. Som resultat registreres inputtet ikke gennem en documentListener, men som et KeyEvent. De indtastede informationer samles i en streng, og dette anvendes som præfiks. Derudover forholder programmet sig ikke til, hvad der sker, hvis brugeren sletter noget i midten af søgerstrengen. Derfor slettes hele brugers input hvis backspace anvendes, istedet for blot at slette et enkelt bogstav. Slutteligt rettes det første indtastede bogstav til et stort bogstav, hvis dette ikke er tilfældet, da adresserne i søgertrien står med stort startbogstav.

4.8 Opsummering

Vi har opnået en brugergrænseflade med et modulært design gennem en række byggeklasser til alle komponenter. Til at behandle kortdata har vi brugt en OSMHandler, der gør det muligt at segmentere data afhængig af, hvilken geografisk information det repræsenterer. Kortvisningen er foretaget på et canvas, der understøtter diverse tegneoperationer. Slutteligt har programmets store ressourceforbrug gjort det nødvendigt at bruge effektive datastrukturer (quadtræ) og søgeralgoritmer (Dijkstra) til at lagring og søgning af kortdata.

Kapitel 5

Teknisk beskrivelse

5.1 Struktur - UML diagrams

Unified Modeling Language (UML) er en standard for udseende af diagrammer til beskrivelse af strukturer og forløb i objekt-orienterede softwaresystemer. Formålet med UML er grafisk at vise sammenhængen mellem de forskellige klasser. Der findes to former for UML diagrammer: Struktur diagrammer og adfærdsdiagrammer.

5.2 Strukturdiagrammer

Strukturdiagrammerne beskriver, som navnet siger, strukturen i et system. Til gengæld viser de ikke noget om systemets adfærd. Indenfor struktur diagrammer kan man lave klasse diagrammer og pakkediagrammer.

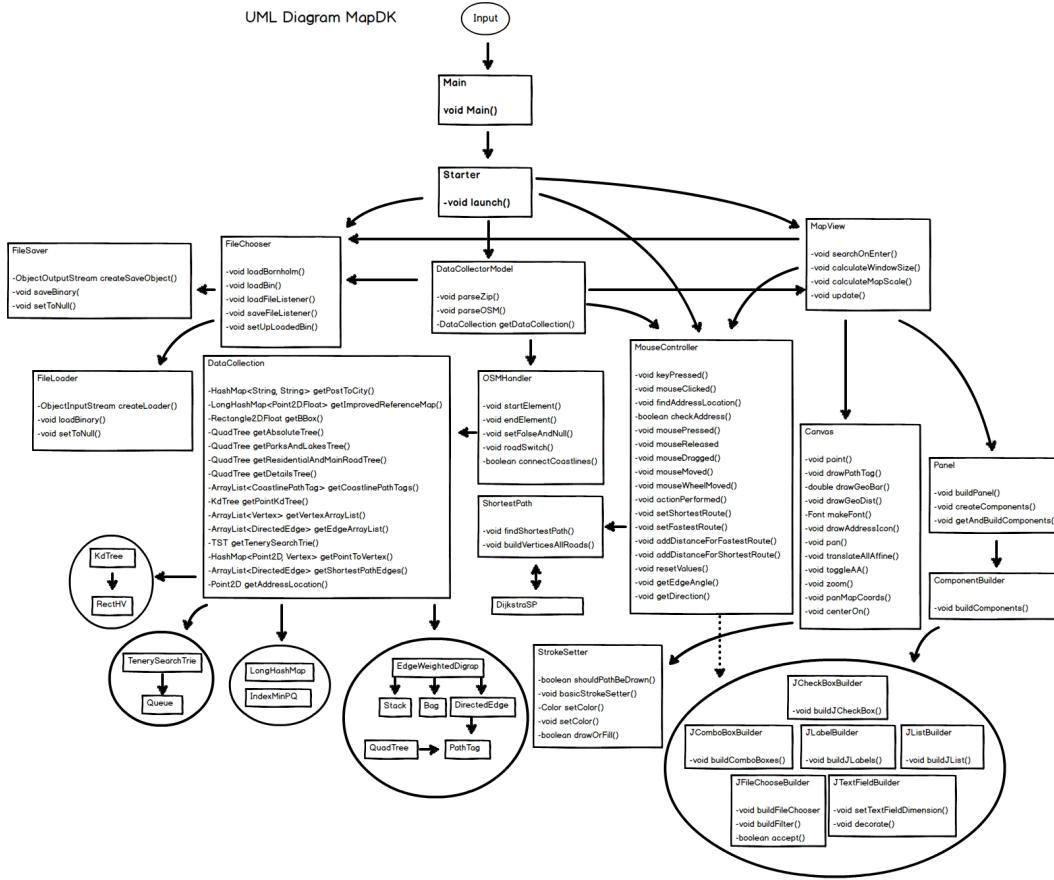
5.2.1 Pakkediagrammer

Pakkediagrammer viser kodens opdeling i pakker. Disse pakker indeholder de klasser der i fællesskab udfører bestemte slags opgaver. Vores pakkeopdeling af lavet udfra MVC mønsteret, og vi har derfor en pakke med Controller-klasser, en med Model-klasser og en med View-klasser.

5.2.2 Klassediagrammer

KLassediagram definerer en måde at beskrive alle eller udvalgte dele af et objektorienteret systems klasser. Diagrammet viser strukturen mellem de udvalgte klasser, f.eks.

- Arv: Hvis en klasse er subklasse eller implementerer et interface (grænseflade-klasse).
- Aggregering: Hvis et objekt eksisterer på grund af et andet objekt. Hvis det overordnede objekt slettes, slettes de underordnede objekter også.
- Afhængighed: Andre tilfælde, hvor en klasse er afhængig af kendskabet til en anden klasse, i praksis ved at den anden klasse opträder som parameter i en metode eller en lokal variabel.



Figur 5.1: UML diagram over klassernes sammenhæng

Klasserne og deres indbyrdes forhold er repræsenteret gennem UML-diagrammet. Den viser de enkelte klasser, samt deres hovedmetoder. Metoder som tilgår eller sætter noget er i de fleste tilfælde frasorteret, medmindre de har haft et særligt formål.

UML diagrammet er bygget op efter, hvilke klasser der skaber de andre klasser, og hvilke klasser som bruges i deres konstruktør.

Der er samlet to grupperinger af klasser; vores JComponent byggeklasser samt datastrukturer. Byggeklasserne er samlet, da de alle er af en mindre størrelse og bliver oprettet under samme hovedklasse, `ComponentBuilder`.

Datastrukturerne er samlet, da deres formål er ens data lagres i `DataCollection`-klassen. Yderligere er metoderne ikke vist i Datastruktur klasserne. De er fravalgt, idet klasserne har mange interne metoder og deres funktionalitet fremkommer gennem andre klasser.

Den stiplede linje fra `MouseController` til gruppen af JComponents indikerer, at `MouseController` registrerer ændringer foretaget i de givne komponenter.

Kapitel 6

Testing

Formålet med testing er først og fremmest at finde bugs. Dette sammenkoblet med interessen i at observere, om ens program rent faktisk virker og gør som ønsket (funktionalitetstest). Det bør ikke forveksles med et ønske om at finde alle fejl eller at vise at programmet er korrekt. Testen lever først op til formålet, når en fejl findes (destruktiv) og ikke når testen ikke fejler (konstruktiv). Hvis testen ikke finder fejl, så er den delvis succesfuld, da den viser at et udsnit af programmet virker, men samtidig er dette bundet op af en erkendelse om, at et program altid besidder fejl, fordi der findes uendelig input. Testing er nemlig en uendelig proces og bygger på en generel antagelse om, at der optræder bugs.

6.1 Black Box

Black box testing tager udgangspunkt i specifikationen og det programmet skal kunne. Her genererer man altså test cases før kodningen. Om programmet fungerer, afhænger af hvad man forventer ifølge specifikationen [Sestoft].

6.2 Address Parser

Brugeren har i programmet mulighed for at søge på en given adresse på kortet. For at sikre at håndteringen af dette brugerinput virker som det skal, har vi skrevet nogle test cases, som efterprøver den fundationale `parse()` metode i vores `AddressParser` klasse. Vi har taget afsæt i unit tests, hvor forskellig input afprøves selvstændigt. Hver test case tester et scenario som er passende ift, hvad man kunne tænke sig en klient ville gøre ved søgning på en adresse. Adressen kan i vores tilfælde enten være korrekt eller forkert afhængig af nogle regler for, hvordan en gyldig adresse ser ud i Danmark. Dette sammenkobles med det forventede resultat. Vi tester altså primært parseren for, om den håndterer adresserne rigtig, hvadenten der er tale om en gyldig adresse eller tale om en adresse, som er ugyldig og derfor ikke bør sendes videre. Se eksempel 1 under "AddressParser"-i afsnittet "kildekode" under bilag for illustration af en given test case. OSM filen indeholder ikke informationer omkring de forskellige adressers etager, herunder heller ikke hvorvidt lejligheden ligger th, tv eller mf. Dette betyder at `AddressParser` skal parse informationen omkring den gældende adresse, og herefter ignorere etage og side.

6.2.1 Definitionen på en adresse

Først og fremmest bør medtages en definition på, hvordan en adresse er opbygget i en dansk kontekst, da dette er fundamentet for det input vi tester i parseren. En adresse består af et vejnavn, husnummer, postnummer, by og evt. etage og dørnummer, hvis der er tale om en lejlighedskompleks. Indimellem disse informationer kan der optræde et vilkårligt antal mellemrum eller kommaer for at adskille tingene og gøre adressen mere overskuelig. Eksempelvis er der som regel komma eller mellemrum mellem vejnavn og husnumre. Disse overvejelser står også beskrevet i et kommentarfelt øverst i kildeteksten for 'AddressParser' klassen.

6.2.2 Test cases

I og med testing er en uendelig proces har vi måtte prioritere, hvilke test cases der var vigtigst. Denne vurdering baserer sig primært på et ønske om at dække de scenarier, som er mest tænkelige i brugen af vores program hos en almen dansker. Samtidig bygger vores test cases på en antagelse om, at hvad der gælder for ét typetilfælde også gælder for alle andre tilfælde af samme type. Eksempelvis generaliserer vi ud fra en test på vejnavne, at testen virker for alle vejnavne i Danmark.

6.2.3 Ækvivalensklasser

Teoretisk omtales dette som undergrupperinger af ækvivalensklasser. Vi har altså lavet en test case for hver kategori af input (ækvivalensklasse). Hvis der opstår en fejl, antager vi at der er fejl i alle input indenfor ækvivalensklassen. Omvendt antager vi at programmet virker for alle input indenfor ækvivalensklassen, hvis vores test passerer med ét input derfra. I vores tilfælde tilsvarer disse kategorier af input måden man kan sammensætte en adresse på.

6.2.4 Vejnavn

Vores test cases bliver brugt i kronologisk rækkefølge alt efter, hvordan en adresse starter og slutter. Det første isolerede scenarie vi tester er derfor, om vores parser godtager danske vejnavne (se `testStreet()`). I dette henseende er det vigtigt at afprøve vejnavne, som også inkluderer bogstaver såsom æ, ø og å, der er unikke for det danske lexicon. Samtidig er det vigtigt at medtage muligheden for vejnavne bestående af flere ord, hvilket vi også dækker i separat test case, `testMultiStreet()`.

6.2.5 Husnummer

I vores test, `testStreetAndNumber()`, tjekker vi om parseren godtager et vejnavn efterfulgt af et husnummer. Husnummeret bør ifølge boligregisteret begynde med 001 og slutte med 999 i ulige vejsider, så vi bør i dette tilfælde tjekke parseren omkring grænsetilfældene. Eksempelvis kunne man tjekke om den godtager 0, 1000 og fx. 5, så man ved den virker indenfor det gyldige interval og håndterer forkert input såsom 0 og 1000 korrekt. Det kunne tænkes at klienten fik en fejlmeddelelse ved indtastning af en adresse med et ugyldigt husnummer, og blev bedt om at prøve igen. Uover dette har vi også udarbejdet nogle tilhørende test cases, der tjekker om parseren modtager husnumre med bogstaver i såsom 5A. Dertil også at den kun gör det for et suffix (etterstavelse) og ikke et præfix (forstavelse).

6.3 Andre testklasser

Følgende testklasser bliver brugt i programmet som blackbox testing; TestTST, OSMHandlerTest, CalculatorTest, DataCollectionModelTest, DijkstraTest, AddressParserTest og QuadTreeTest. Bemærk at QuadTreeTest komplementerer og beviser dele af white box testen af quadtræet. Programmet indeholder stadig nogle testklasser, som ikke er blevet fjernet på trods af manglende brug. Eksempelvis skulle ViewBoxTest være blevet brugt til at teste, om der kun bliver tegnet elementer fra en skræddersyet osm testfil indenfor det synlige tegneområde. Der bør dog ses bort fra denne klasse, da den forkerte osmfil i mappen "TestMaps" er blevet brugt.

6.3.1 TestTST

TestTST klassen tester om de gængse metoder for en præfiksbasert søgetrige virker. TST klassen svarer til en symbol tabel med nøgle-værdi par, der understøtter typiske metoder såsom put, get, contains m.m og bruges til at finde adresser, der starter med bestemt præfiks. Testklassen tjekker med nogle testinput i en queue, om klassen finder de rigtige adresser ved søgning efter bestemt præfiks og indeholder de rigtige adresser efter brug af søgetriens gængse metoder.

6.3.2 OSMHandlerTest

Denne testklasse sikrer at programmet fejler, hvis osm-filen med kortdata er tom, korrupt eller indeholder misvisende data. For at teste håndteringen af sådanne filer har vi udarbejdet filer med korrekt og forkert data i mappen TestMaps.

6.3.3 CalculatorTest

Calculator testklassen tjekker om kortets målestokksforhold bliver udregnet korrekt i kortets afstandsmåler. Afstandsmåleren tilsvarer "Geobaen" i programmet, der viser hvad et givent stykke på kortet svarer til i henholdsvis kilometer og meterafstand. Testklassen tjekker helt specifikt om afstanden ligger indenfor et acceptabelt niveau med en afvigelse på højst 0,01.

6.3.4 DataCollectionModelTest

Denne testklasse tilføjer data til lister ud fra forskellige attributter, som tilsvarer dem fra en osm-fil med kortdata.

6.3.5 DijkstraTest

Dijkstra testklassen skaber først en graf med en forventet korteste rute og tjekker, om programmets implementation af algoritmen finder en rute og sørger for, at det er den kortest mulige.

6.4 TestMaps

Mappen TestMaps indeholder skræddersyede osm-filer, der er lavet med henblik på at forsimple og indsætte testscenarierne. Istedet for at teste om programmet håndterer kortdata korrekt, finder den korteste rute, tegner det rigtige antal elementer m.m. for en stor osm-fil, bruger vi kun en fil med et lille udsnit kortdata.

6.5 Test Fixtures

På trods af klare kodekonventioner har vi haft udfordringer med høj kobling mellem de klasser, der bruges ekstensivt i programmet (MouseController, DataCollector, Canvas, OSMHandler m.m.). Som resultat har vi været nødsaget til at bruge test fixtures, der sikrer at tests bliver evalueret i et virkelighedsnært miljø. En test fixture svarer til en fikseret objekttilstand for en given klasse. Eksempelvis har vi en eksakt kopi af DataCollector i vores Test mappe. Formålet med en test fixture er at sikre et afgrænsset testmiljø, hvor ens testresultater kan reproduceres. Fixture klasserne i programmet bliver udelukkende brugt, fordi mange af disse klasser afhænger af hinanden (jf. Class Dependancies). Klasser bruger hinanden indbyrdes og er koblet i en sådan grad, at en given klasse A ikke kan genbruges uden adgang til en anden klasse B. Dette er taget op til ydeligere reflektion i afsnittet ”Videre udvikling”.

6.6 White Box

White box testing bygger på et eksisterende kendskab til koden, og man forsøger derfor at teste implementeringen som den er på tidspunktet. Det er derfor først muligt at lave white box tests, når man har koden. Samtidig påpeger denne type tests ikke alle fejl, da den eksempelvis ikke tester for scenarier, som ikke er medregnet i den eksisterende kode. Eksempelvis tester vi ikke vores parser for alle typer input og måder at opstille addresser på. Dertil kunne vi udvikle tests på baggrund af nye funktionaliteter, som programmet skal opfylde ifølge kravsspecifikationen (black box).

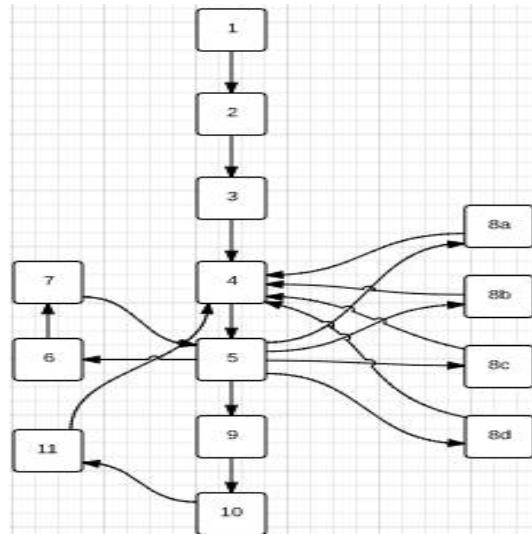
6.6.1 Quadtræ - insertion metode

Quadtræet er en af de vigtigste datastrukturer i programmet, da det er ansvarlig for holde på alle elementer, der skal tegnes på kortet. Derfor er det relevant at whitebox-teste træets insertion-metode, idet den afgør, om de geografiske informationer bliver indsat korrekt. Testen gør brug af det samme rektangel og PathTag objekter, som bruges i test-klassen: QuadTreeTest. Med undtagelse af ”path-Tag5”, der i stedet refereres ”pathTag6” i denne test. Se bilagene for Coverage table og figurer over rektangler, hvori PathTag objekter indsættes. Metodens kontrolflow er vist og beskrevet nedenunder.

Kontrolflow

1. Insert metoden (PathTag pathTag) - START
2. Check om Path2D eller PathTag objektet er valid. Hvis ja fortsæt til 3, ellers exception.
3. Er Path2D objektet indenfor rod-rektanglet. Hvis ja fortsæt til 4. Ellers stop her.
4. Insert-metoden (PathTag pathTag, Node node) Gå til punkt 5
5. Er knudepunktet (her roden) firdelt? Gå til kontrolpunkt 6 for svar. Hvis nej gå til kontrolpunkt 9. Hvis ja, gå til 8a og/eller 8b og/eller 8c og/eller 8d, der herefter rekursivt går tilbage til kontrolpunkt 4, indtil at alle knudepunkter bliver sendt til kontrolpunkt 9 og stopper.
6. isQuaded(node n) Gå til punkt 7

7. Tjek om roden eller knudepunktet indeholder 4 underliggende rektangler. Hvis ja, evalueres det boolske udtryk sandt. Ellers falsk. Retuner til punkt 5 med svaret og gå til kontrolpunkterne i 8 eller kontrolpunkt 9.
8. Controlpunkt
 - (a) Tjekker om elementet er inden for det nordvestlige rektangel. Hvis ja, gå tilbage til kontrolpunkt 3 med denne knude (subtræ). Ellers gør intet.
 - (b) Tjekker om elementet er inden for det nordøstlige rektangel. Hvis ja, gå tilbage til kontrolpunkt 3 med denne knude (subtræ). Ellers gør intet.
 - (c) Tjekker om elementet er indenfor det sydvestlige rektangel. Hvis ja, gå tilbage til kontrolpunkt 3 med denne knude (subtræ). Ellers gør intet.
 - (d) Tjekker om elementet er indenfor det sydøstlige rektangel. Hvis ja, gå tilbage til kontrolpunkt 3 med denne knude (subtræ). Ellers gør intet.
9. Indsæt elementet, og hvis grænsen på det maksimale antal elementer i roden eller knuden overstiges, gå til kontrolpunkt 10, hvorfra rektanglet firdeles til 4 nye subtræer (knuder) med hver sit rektangel. Ellers STOP.
10. quad(node n) Del knudens rektangel op i 4 subtræer med hver sit rektangel, og gå til kontrolpunkt 11
11. Gå til kontrolpunkt 4 x antal gange, hvor x er det maksimale antal af elementer, der må optræde i hvert knudepunkt. Hertil sendes også den pågældende knude, som er blevet firdelt, sammen med PathTag-objekterne fra knuden.



Figur 6.1: Kontrol-flow af insertion method

Branchdækning

Whiteboks testen formår at opnå “branch”-dækning, eftersom hver opførelse af kontrol fra kontrolpunkt til kontrolpunkt bliver undersøgt af testen, og det forventede samt aktuelle resultat bliver beskrevet. Med undtagelse af kontrolpunkt 9, hvor det ikke er muligt at udføre nul iterationer. Dette skyldes at programmet først firdeler data ydeligere, når grænsen for det maksimale antal elementer overstiges, og minimum for programmet vil altid være 1. Whitebox testen afslører ligesom blackbox, at der ikke umiddelbart er noget galt med insertion metoden. Medmindre datagrænse er sat til 1, hvorved programmet firdeler og indsætter data for evigt, hvilket ikke blev opdaget af blackbox testen. Whitebox testen er dog fortsat med til at forstærke vores bekræftelse om, at metoden opfører sig, som den skal. Slutteligt er den med til at komplementere black box testen og sikre vores tillid til, at data bliver håndteret og vist korrekt på kortet.

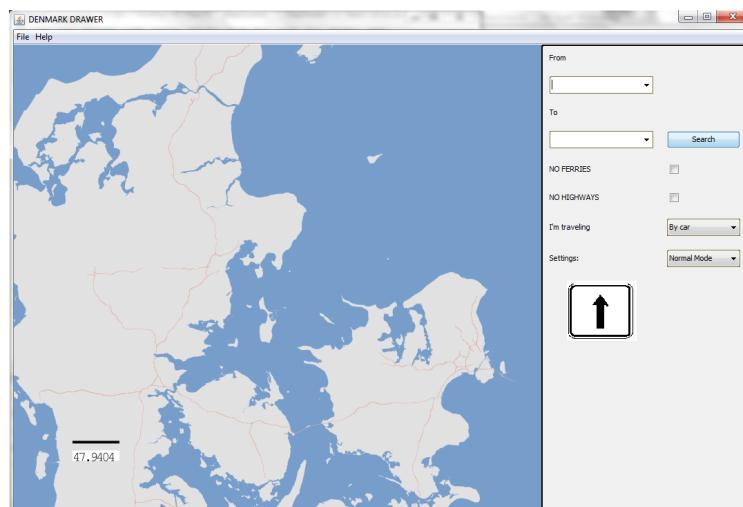
6.7 Opsummering

Vi har gennem forløbet fået et indgående kendskab til både black- og white box testing. AdresseParseren er blevet testet ud fra dens specifikation, der afgør hvordan adresseinput bør håndteres. For at imødekomme denne testing har vi opstillet forskellige scenarier for, hvilke adresser der bør godtages og omvendt håndteres. Slutteligt er white box testing blevet benyttet på et senere tidspunkt til at teste implementeringen for vores quadtræ.

Kapitel 7

Brugerguide

På billedet nedenfor ses brugergrænsefladen for programmet. Til højre er der et panel med en række funktioner til interaktion med kortet. Ved at skrive i det øverste søgefelt og trykke på den øverste knap 'Find Address' markeres adressen med en rød lokationsmarkør på kortet, såfremt adressen eksisterer. Udfyldes adressefeltet i både det øverste og nederste søgefelt, kan 'Route' knappen anvendes til at vise en rute mellem adresserne, hvor den blå og røde markør tilsvarer henholdsvis start- og slutpunkt.



Figur 7.1: Åbnings vinduet

Nedenunder findes tre tjekbokse. Ved at klikke i den øverste, kan man undgå motorveje på ruter, såfremt man fx hellere vil køre ad mindre veje. De andre bokse anvendes til at vælge mellem den korteste og hurtigste rute. Den hurtigste rute tager højde for fartgrænser på vejene ved rutesøgning og vil derfor opprioritere motorveje. Bemærk at kun én af mulighederne kan slås til. Under tjekboksene eksisterer der to rullemenuer. I den øverste menu kan angives det anvendte transportmiddel. Bemærk at stier og små veje ignoreres, hvis man er i bil og motorveje ignoreres, hvis man er på cykel eller gåben. Den nederste rullemenu tillader brugeren at ændre farverne i tilfælde af farveblindhed. Under de to rullemenuer ses et vises fejlmeddelelser og instruktioner til ruten. Allernederst i panelet kan brugeren se længde- og breddegrader for musens placering på kortet, samt den vej der ligger nærmest på musen. Slutteligt kan brugeren indlæse og gemme kort i menubjælken under 'File' og søge hjælp under 'Help'.

Kapitel 8

Konklusion

8.1 Videre udvikling

8.1.1 Høj kobling

I forbindelse med afslutningen på Førsteårsprojektet er der nogle områder, som bør tages op til genovervejelse ved videreudvikling af programmet. Den mest iøjefaldende udfordring ved projekts afslutning har været programmets høje kobling imellem nogle af klasserne. Som konsekvens har det været svært at teste programmets implementering, og vi har været nødsaget til at kopiere objekttilstanden fra visse klasser over i test fixtures for overhovedet at kunne dække forskellige testscenarier. Et konkret eksempel på en klasse, der er højt koblet med andre klasser er vores instans af DataCollectorModel. Objektet holder på alt kortdata fra OSM-filen og bliver brugt som parameter i vores View, Controller og FileChooser. For at løsne klassekoblingen kunne dette objekt have afhængt af en abstraktion istedet for en konkret type. Det konkrete objekt og dets tilstand er meget eksponeret, hvorimod et interface ikke holder på nogen ”bestemt adfærd”. Selvom vi løbende ændrer implementeringen, vil det ikke påvirke andre klasser, som implementerer det. DataCollectorModel kan i dette henseende ses som et ”gudeobjekt”, der ved for meget. Vi kun har netop ét instans af klassen, som holder på alt data og det skal kaldes alle andre steder. Alternativt kunne man bruge nogle interfaces eller referencer for at mindske koblingen. Selvom det ikke er meget tydeligt ud fra vores UML diagram, så bliver DataCollection-klassen hentet rigtigt mange gange via getDataCollection() fra DataCollectorModel.

8.1.2 Rutevejledning

Et andet interessant emne at genoptage ved videre udvikling relaterer sig til rutevejledningen. Rutevejledningen kan opdateres og blive mere præcis ved at justere spektret, der udregner retninger på kortet. Eksempelvis bør spektret sættes ud fra forgående retninger man har kørt på, så sving bliver mere præcise. Derudover bør rutevejledningen også tage højde for den tidligere nævnte problemstilling, hvor veje skifter navn på en lige strækning for derefter at dreje. Grunden til at programmets rutevejledning kun tager højde for sving skyldes at mange veje har naturlige sving som opstår ved skifte i vejnavne, der er overflødige at medtage i rutevejledningen. Dette er en forholdsvis kompleks problemstilling og derfor valgte gruppen at sætte skællet ved skift af vejnavn. Sluteligt bør det medtages, at programmet ikke tager højde for ensrettede veje. Som konsekvens vil den hurtigste rute risikere at blive udregnet via modgående kørselsbaner.

8.1.3 AdressParser

Nævneværdigt er også klassen Address Parser, der i den endelige version af programmet endte med at blive boykottet. Den blev brugt til at validere adresserne direkte i OSMHandler klassen, hvilket resulterede i at den konsekvent tjekkede alle adresser for en given fil brugt i programmet. Til videre udvikling burde den først blive brugt til at filtrere fejlbehæftede adresser fra, når brugeren søger på en given adresse, så den ikke skal gøre det for alle adresser for hver gang programmet opstartes.

8.1.4 Klassestruktur

Slutteligt kan programmet fortsat gøres pænere og mere læsevenligt ved at bryde mange af funktionalliteterne op i ydelige klasser og metoder samt bruge nedarvning og interfaces. Et konkret eksempel på en klasse, der bør refaktoreres er OSMHandleren og dens start- og endElement() metoder. Istedet for at behandle alle kortattributter på én gang kunne man separere dem ud i submetoder, der tager bestemte attributter som parameter. Brugergrænsefladen bør også refaktoreres, så alle komponenterne nedarver fra panelet istedet for at skulle bygge alle komponenterne i ComponentBuilder klassen.

8.2 Konklusion

Vi har gennem Førsteårsprojektet erfaret, hvor vigtigt det er at reflektere over ens design og arkitektur i et system, før man påbegynder kodningen. Vi har under hele forløbet forsøgt at gøre os nogle tanker omkring, hvad problemet er, og hvordan vi vil løse det. I gruppen har vi arbejdet iterativt på de problemstillinger, der afstekkom i udviklingen af et kortprogram. Selvom vi ikke har løst alle problemer og dertilhørende scenarier mest hensigtsmæssigt, har vi gennem en mindre perfekt løsning skudt os gradvist tættere på målet ved at løse de vigtigste opgaver først. For at kunne gøre dette har vi brutt det overordnede problem op i simplere delproblemer, der samtidig har givet os indsigt i, hvordan det komplekse problem løses. Dette kommer til udtryk i vores klassestruktur, der er skabt med henblik på at opdele ansvarsområderne mellem flere aktører, som løser det overordnede problem i samspil. Slutteligt har introduktionen til algoritmer og datastrukturer gjort det muligt at imødekomme det omfattende ressourcekrav, som programmet bør efterstille for at imødekomme brugerens behov. Ved at applicere principper fra kurset i algoritmer og brugergrænseflade på et ikke-triviel program har vi således opnået en større viden omkring håndteringen af softwareprojekter, og samtidig er vi alt i alt blevet dygtigere programmører.

Kapitel 9

Reflektion over projektet

Det har for alle gruppemedlemmer været en kæmpe omvæltning at gå fra at have et “perfekt” kendskab til en masse grundprincipper i Java kodning til at skulle applicere disse på et af virkelighedens problemer. Vi har tilegnet os ny viden på flere fronter herunder, hvordan man strukturerer et softwareprojekt i praksis med udgangspunkt i objektorienteret programmering, der kan bruges til at opdele koden i klasser, som håndterer hver sit ansvarsområde. Denne tankegang fik vi introduceret gennem kurset i grundlæggende programmering, som har skærpet vores forståelse for dens virkning i praksis. Samtidig har vi nu, med egne øjne erfaret, hvilke udfordringer der eksisterer i et softwaremiljø, hvor man skal administrere et projekt mellem flere mennesker. Vi fik bekræftet, hvor effektiv versionsstyring er til at administrere et softwareprojekt mellem flere programmører, der skal kunne se hinandens ændringer og arbejde samtidig. Dette er alfa omega, da man ofte arbejder uafhængigt af tid og sted indenfor softwarebranchen. Ved at bruge versionsstyring har vi kunne genoptage arbejdet uden videre og undgå unødvendige distraktioner med risiko for at afbryde arbejdet. Derudover har kravene til programmets ressourceforbrug været en kæmpe udfordring i sig selv. For at imødekomme dette har vi udnyttet vores viden fra kurset i algoritmer om effektive datastrukturer og søgealgoritmer. Kurset har udstyret os med værktøjer til at vurdere, hvilke algoritmer og datastrukturer der er bedst egnede i en bestemt kontekst. Samtidig har den praktiske tilgang gjort det muligt for os at implementere dem selv, så vi også lærer koncepterne indenfor programmering. Arbejdsopgaver er blevet uddelt via applikationen Trello, som er et redskab til projektstyring, der giver en oversigt over hvilke aktiviteter, der skal nås hvornår. Redskabet har i høj grad bidraget til at strukturere og prioritere vores arbejde, hvilket vi har fundet nødvendigt i takt med at skulle nå de ugentlige deadlines. Undervejs er vi desuden blevet introduceret til mange nyere og bedre måder at implementere dele af vores løsninger på, men samtidig har vi været bundet op på en deadline. Selvom det har været svært for os at give slip på nogle ting, har vi accepteret omstændighederne og erkendt, at et program aldrig er fejlfrit og uforanderligt. Man bør derfor være åben overfor de udfordringer, der foreligger både i ens kode, men også i valget mellem det mangfoldige udbud af teknologier, der er målrettet samme problem. I udviklingen af software har vi altså lært at omfavne forandring sammen med den hurtige tilpasning til nye teknologier, der skaber incitament til at prøve nye ting.

Figurer

3.1	Mock-up af GUI	13
3.2	Illustration af farveblindhed	14
4.1	Visualisering af MVC modellen	15
4.2	Udsnit af OSM-fil på en ”node”tag.	19
4.3	De 3 scenarier der kan forekomme, imens kyslinjerne forbinder. Den blå cirkel er startpunktet og den røde cirkel er slutpunktet.	21
4.4	Illustration af quadtræets opdeling af kortet	24
4.5	Illustration af k-d træets datastruktur	26
4.6	Quadtræ firdeler på et rektangel mens kd træ todeler ud fra data henholdsvis lodret og vandret	27
4.7	Kanter oprettes mellem eksisterende knuder	29
4.8	Ny knude og kant til eksisterende knude	29
4.9	Knuder oprettes og forbides med ny kant	29
5.1	UML diagram over klassernes sammenhæng	34
6.1	Kontrol-flow af insertion method	39
7.1	Åbningsvinduet	41

Bibliografi

- ADLAKHA, VAIBHAV. *Serialization: Pros and Cons*. URL: <http://adlakhavaibhav.blogspot.dk/2009/10/serialization-pros-and-cons.html>.
- affiliates., Oracle and/or its. *Class AffineTransform*. URL: <http://docs.oracle.com/javase/7/docs/api/java.awt.geom/AffineTransform.html>.
- . *Serializable JavaDoc*. URL: <http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html>.
- Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. URL: <http://www.w3.org/TR/CSS21/>.
- Chen, Daniel. *algs4/kdtree/KdTree.java*. URL: <https://github.com/dychen/algs4/blob/master/kdtree/KdTree.java>.
- How to draw a (or two) million lines fast?* URL: <https://community.oracle.com/thread/2503301>.
- Jenny, Bernhard. *Color Oracle*. URL: <http://colororacle.org/> (sidst set 2013).
- Lounge, GIS. *Making Color Blind Friendly Maps*. URL: <http://www.gislounge.com/making-color-blind-friendly-maps/>.
- Ltd, Envato Pty. *MVC for Noobs*. URL: https://learnit.itu.dk/pluginfile.php/113876/mod_resource/content/0/GRPRO-08.pdf (sidst set 2014).
- Maps, Open Street. *Open Street Maps*. URL: <https://www.openstreetmap.org/#map=17/55.17518/14.92136> (sidst set 2015).
- Martin, Robert C. (2009). *Clean Code*. Pearson Education, Inc. URL: http://www.polyteknisk.dk/related_materials/9780132350884_Sample.pdf.
- Overflow, Stack. “no valid constructor” when serializing a subclass of Path2D.Double. URL: <http://stackoverflow.com/questions/10472508/no-valid-constructor-when-serializing-a-subclass-of-path2d-double>.
- RUC. *Algoritmisk geometri*. URL: http://en.wikipedia.org/wiki/Affine_transformation.
- Sedgewick, Robert (2011). *Algorithms*. 4th Edition. Addison-Wesley Professional, Boston. URL: <http://algs4.cs.princeton.edu/home/>.
- Sestoft, Peter. *Systematic Software Testing*. IT University of Copenhagen.
- stackoverflow. “no valid constructor” when serializing a subclass of Path2D.Double. URL: <http://stackoverflow.com/questions/10472508/no-valid-constructor-when-serializing-a-subclass-of-path2d-double>.
- Sørensen, Troels Bjerre. *Danmarkskort: Visualisering, Navigation, Søgning og Ruteplanlægning*. URL: <https://learnit.itu.dk/pluginfile.php/121930/course/section/68144/lec10.pdf> (sidst set 2015).
- TA’s, (2015). *Clean Code: en kort forelæsning om grundprincipperne i Robert C. Martin’s Clean Code*. URL: <https://learnit.itu.dk/pluginfile.php/121930/course/section/68142/cleanCode.pdf> (sidst set 2015).
- univericty, Princeton. *Quadtree.java*. URL: <http://algs4.cs.princeton.edu/92search/QuadTree.java.html>.
- . *RectHV.java*. URL: <http://algs4.cs.princeton.edu/code/RectHV.java.html> (sidst set 2015).

Wikipedia. *Affine transformation*. URL: http://www.akira.ruc.dk/~keld/teaching/algoritmedesign_f08/Slides/pdf/09_Intervalsoegning.pdf.

Wolfram Research, Inc. *Affine Transformation*. URL: <http://mathworld.wolfram.com/AffineTransformation.html>.

Bilag A

Bilag

A.1 Kildekode

A.1.1 Clean Code - Eksempel 1

Følgende eksempel viser et udsnit af koden, der holder på lister med de PathTag objekter vi bruger til at afgøre, hvordan forskellige ting på kortet skal tegnes.

```
ArrayList<PathTag> pathTagstoBeDrawn;  
ArrayList<PathTag> pathTagstoBeDrawn2;  
ArrayList<PathTag> pathTagstoBeDrawn3;  
ArrayList<PathTag> pathTagstoBeDrawn4;
```

Listenavnene ovenfor gengiver rigtig nok, at de holder på PathTag objekter med tegneinformationer. Til gengæld fremgår dette allerede i ArrayList-signaturen og er derfor redundant. Samtidig bliver det eksplisitte navn længere, hvilket også kommer til udtryk i lange metodekald som konsekvens af den meget beskrivende tilgang (en af ulemperne ved clean code). Slutteligt står navnene iterativt og er adskilt med numre istedet for at sige noget om, hvilken kortdata de holder på (fx. veje eller bygninger). Her er en revideret udgave af de det samme kodeudsnit, hvor den eneste forskel er navngivningen.

```
ArrayList<PathTag> highwayPaths;  
ArrayList<PathTag> parkPaths;  
ArrayList<PathTag> smallPaths;  
ArrayList<PathTag> mainroadPaths;
```

Navnene forsøger nu istedet at afspejle, hvad der tegnes på kortet med den data listen holder på. Samtidig er præfikset ”pathTags” blevet fjernet til fordel for en mere kompakt navngivning, der afspejler formålet med listen. Eksempelvis bliver den første liste brugt til at holde på data for linjesegmenter tilknyttet motorveje. Som resultat er det nu nemmere for en ekstern programmør at reviewe, debugge og ændre programmet i fremtiden, fordi vedkommende har kendskab til, hvad der sker med listerne under tegneprocessen på kortet.

A.1.2 Clean Code - Eksempel 2

```

private void buildButton()
{
    addressSearchButton = new JButton("Find Address");
    setButtonDimension(addressSearchButton);

    routeSearchButton = new JButton("Route");
    setButtonDimension(routeSearchButton);
}

private void setButtonDimension(JButton button)
{
    button.setPreferredSize(new Dimension(80,25));
    button.setMaximumSize(new Dimension(80, 25));
    button.setMinimumSize(new Dimension(80, 25));
}

```

A.1.3 Clean Code - Eksempel 3

Følgende kodeudsnit er svært at forstå, fordi de lokale variabler enten ikke er placeret ordentligt eller er navngivet med bogstaver istedet for navneord, der afspejler det bagvedliggende formål.

```

if(!dataCollectorModel.getDataCollection().getShortestPathEdges.isEmpty())
{
    g.setStroke(StrokeSetter.setStroke(Tag.ROUTE.getValue()));
    g.setColor(StrokeSetter.setColor(Tag.ROUTE.getValue()));

    for(int i = dataCollectorModel.getDataCollection().getShortestPathEdges.size()-1; 0 < i; i--)
    {
        DirectedEdge d = dataCollectorModel.getDataCollection().getShortestPathEdges().get(i);
        g.draw(d.getPathTag().getPath2D());
    }
}

```

Helt specifikt er for-løkken meget lang og den lokale variabel ”d”er umulig at forstå for en udefrakommende, der læser kode, hvori variablen bliver brugt. Forståelse for brugen af ”d”forudsætter, at man på forhånd ved, hvor den bliver defineret og i hvilken kontekst den indgår, hvilket primært gælder for programmets oprindelige udviklere. Koden nedenfor er nu blevet revideret, så for-løkken er blevet mere kompakt, den intetsigende variabel ”d”afspejler sit indhold og koden er overordnet blevet mere selvforklarende.

```

if(!dataCollectorModel.getDataCollection().getShortestPathEdges.isEmpty())
{
    g.setStroke(StrokeSetter.setStroke(Tag.ROUTE.getValue()));
    g.setColor(StrokeSetter.setColor(Tag.ROUTE.getValue()));
    int shortestPathEdgeSize = dataCollectorModel.getDataCollection().getShortestPathEdges.size()-1;
    for(int i = shortestPathEdgeSize; i > 0; i--)
    {
        DirectedEdge directedEdge = dataCollectorModel.getDataCollection().getShortestPathEdges().get(i);
        g.draw(directedEdge.getPathTag().getPath2D());
    }
}

```

Den initialiserede variabel i løkkens (forrige side) bliver sat til samme værdi som den lokale variabel ”shortestPathEdgeSize”, der forklarer en, at den holder på antallet af kanter for en given korteste rute. Variablen ”directedEdge” er blevet navngivet af hensyn til læsevenligheden.

A.1.4 Canvas klassen - Tags Array og tegneproces

Arrayet med Tags ses forneden og kan findes i kildekoden på linje 31-32. Tegneprocessen foregår på linje 113-130.

```

private int[] tags = {
    Tag.COASTLINE.getValue(), Tag.PARK.getValue(), Tag.LAKE.getValue(),
    Tag.WATERWAY.getValue(), Tag.BIKE.getValue(), Tag.PATH.getValue(),
    Tag.RESIDENTIAL.getValue(), Tag.MAINROAD.getValue(), Tag.HIGHWAY.getValue()
};

for(int i = 0; i < tags.length; i++)
{
    int tag = tags[i];

    if (StrokeSetter.shouldPathBeDrawn(tag, trueZoomScale))
    {
        g.setStroke(StrokeSetter.getStrokeFromTag(tag));
        g.setColor(StrokeSetter.setColor(tag));
        boolean drawOrFill = StrokeSetter.drawOrFill(tag);

        if (tag == Tag.HIGHWAY.getValue() || tag == Tag.COASTLINE.getValue()) {drawPathTag(pathTagsToBeDrawn, tag, qTreeView, g, drawOrFill);
        else if (tag == Tag.PARK.getValue() || tag == Tag.LAKE.getValue() ||
        tag == Tag.WATERWAY.getValue()) {drawPathTag(pathTagsToBeDrawn2, tag, qTreeView, g, drawOrFill); }
        else if (tag == Tag.RESIDENTIAL.getValue() ||
        tag == Tag.MAINROAD.getValue()) {drawPathTag(pathTagsToBeDrawn3, tag, qTreeView, g, drawOrFill); }
        else if (tag == Tag.ROUTE.getValue() || tag == Tag.BIKE.getValue()) {drawPathTag(pathTagsToBeDrawn4, tag, qTreeView, g, drawOrFill); }
    }
}

```

A.1.5 OSMHandler - Eksempel 1

A.1.6 OSMHandler - Eksempel 2

```
else if (qName.equals("nd"))
{
    coord = null;
    long id = Long.parseLong(attrs.getValue("ref")); // Reference to prior coordinate
    coord = dataCollection.getImprovedReferenceMap().get(id);

    if (coord == null) return; // Unknown coordinates are ignored
    if (startPosition == null){startPosition = id;} // Coast line start point
    endPosition = id; //Sets the endPosition for Coastlines
    smallWays.add(id);

    if (way == null)
    {
        way = new PathTag();
        way.setPath2D(new Path2D.Float());
        way.getPath2D().moveTo(coord.getX(), coord.getY());
    }
    else way.getPath2D().lineTo(coord.getX(), coord.getY()); // Start on a new road and reset content
}
```

A.1.7 AddressParser - Eksempel 1

Koden nedenfor er testen `testFloorAndSideWithPostcode()` fra AddressParser klassen.

```
@Test
public void testFloorAndSideWithPostcode()
{
    AddressParser[] addresses = new AddressParser[] {
        AddressParser.parse("Holmegårdsvej 5A 2th. 2920 København"),
        AddressParser.parse("Holmegårdsvej 5A 2 th. 2920 København"),
        AddressParser.parse("Holmegårdsvej 5A 2 t.h. 2920 København"),
        AddressParser.parse("Holmegårdsvej 5A 2. t.h. 2920 København")
    };

    for(AddressParser address: addresses)
    {
        assertNotNull(address);
        assertEquals("Holmegårdsvej", address.street());
        assertEquals("5A", address.house());
        assertEquals("København", address.city());
        assertEquals("2920", address.postcode());
    }
}
```

A.1.8 Serializable - Eksempel 1

```
public void saveBinary(String filename)
{
    ObjectOutputStream output = null;
    try
    {
        output = createSaveObject(filename);
        output.writeObject(quadTree);
        output.close();
    }
    catch (FileNotFoundException | IOException e) {e.printStackTrace();}
}
```

A.1.9 Serializable - Eksempel 2

```
public void loadBinary(String filename)
{
    try
    {
        ObjectInputStream input = createLoader(filename);
        quadTree = (QuadTree) input.readObject();
        input.close();
    }
    catch (IOException | ClassNotFoundException e) {throw new RuntimeException();}
}
```

A.2 Whitebox test af insertion i quadtræ

A.2.1 Coverage tabel over alle kontrolpunkter - del 1

Insertion

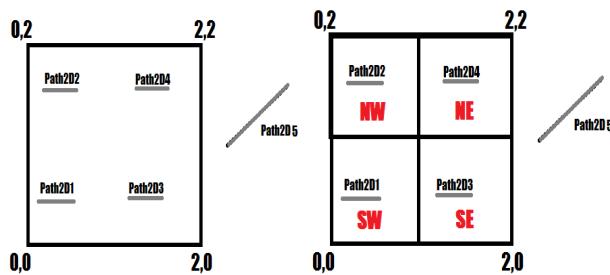
Choice	Input property	Input data set	Path
2. Throw exception	Null	A	1, 2
2. No exception throw	Exactly one	B	1,2, Etc.
3. False	Exactly one outside	C	1, 2, 3
3. True	Exactly one inside	D	1, 2, 3, Etc.
5. False	Number of elements < Limit of node	E	(1,2,3,4,5,6,7,5,9)* Limit - 1 times
5. True	Number of elements >= Limit of node	F	(1,2,3,4,5,6,7,5,9)* Limit, 10, 11,4,5,6,7,5, etc.
7. False	Number of elements < Limit of node	E	(1,2,3,4,5,6,7,5,9)* Limit - 1 times
7. True	Number of elements >= Limit of node	F	(1,2,3,4,5,6,7,5,9)* Limit,10, 11,4,5,6,7,5, etc.
8a. False	No intersection with rectangle (Given that the root has been quadet)	F + H,I,J	(1,2,3,4,5,6,7,5,9)* Limit, 10, 11,4,5,6,7,5,8a,8b,4, 5,6,7,5,9,8c,4,5,6,7,5 ,9,8d,4,5,6,7,5,9
8a. True	Intersection with rectangle (Given that the root has been quadet)	G	(1,2,3,4,5,6,7,5,9)* Limit, 10,11,4,5,6,7,5,8a,4, 5,6,7,5,9,8b,8c,8d
8b. False	No intersection with rectangle (Given that the root has been quadet)	F +G,I,J	(1,2,3,4,5,6,7,5,9)* Limit, 10,11,4,5,6,7,5,8a,4, 5,6,7,5,9,8b,8c,4,5,6, 7,5,9, 8d,4,5,6,7,5,9

A.2.2 Coverage tabel over alle kontrolpunkter - del 2

8b. True	Intersection with rectangle (Given that the root has been quadet)	H	(1,2,3,4,5,6,7,5,9)* Limit, 10,11,4,5,6,7,5,8a,8b , ,4,5,6,7,5,9,8c,8d
8c. False	No intersection with rectangle (Given that the root has been quadet)	F + G,H,J	(1,2,3,4,5,6,7,5,9)* Limit, 10,11,4,5,6,7,5,8a,8b,4,5,6,7,5,9,7,8c,8d,4,5,6,7,5,9
8c. True	Intersection with rectangle (Given that the root has been quadet)	I	(1,2,3,4,5,6,7,5,9)* Limit, 10,11,4,5,6,7,5,8a,8b,8c,4,5,6,7,5,9,8d
8d. False	No intersection with rectangle (Given that the root has been quadet)	F + G,H,I	(1,2,3,4,5,6,7,5,9)* Limit, 10,11,4,5,6,7,5,8a,4,5,6,7,5,9,8b,4,5,6,7,5,9,8c,4,5,6,7,5,9,8d
8d. True	Intersection with rectangle (Given that the root has been quadet)	J	(1,2,3,4,5,6,7,5,9)* Limit, 10,11,4,5,6,7,5,8a,8b,8c,8d,4,5,6,7,5,9
9. False	Number of inserted elements < limit of tree	E	(1,2,3,4,5,6,7,5,9)* Limit - 1 times
9. True	Number of inserted elements == limit of tree	F	(1,2,3,4,5,6,7,5,9)* Limit, etc.
11. Zero iterations	-	-	-
11. One iteration	(Requires that the limit of the tree is: 1) Exactly one iteration	K	1,2,3,4,5,6,7,5,9,10,11,4,5,6,7,5,6c(4,5,6,7,5,8,9,10,11,4,5,6,7,5,8?, etc)*∞
11. More than one iterations	More than one iteration	F	(1,2,3,4,5,6,7,5,9)* Limit, 8,9,3,4,5,4,8a,3,4,5,4,7,8b,3,4,5,4,7,8c,3,4,5,4,7,8d,3,4,5,4,7

A.2.3 Input datasæt med forventede og aktuelle resultater

Data set	Input contents	Expected output	Actual output
A	Null	AssertionError	AssertionError
B	Path2D1	No exception	No exception
C	Path2D5	Size = 0 / False	Size = 0 / False
D	Path2D1	Size = 1 / True	Size = 1 / True
E	Path2D1 + Path2D2 + Path2D3	Size = 3 / False / False	Size = 3 / False / False
F	Path2D1 + Path2D2 + Path2D3 + Path2D4	Size = 0 / True / True	Size = 0 / True / True
G	Path2D2	SW.size = 2 / True / False for rest	SW.size = 2 / True / False for rest
H	Path2D4	NW.size = 2 / True / False for rest	NW.size = 2 / True / False for rest
I	Path2D1	SE.size = 2 / True / False for rest	SE.size = 2 / True / False for rest
J	Path2D3	NE.size = 2 / True / False for rest	NE.size = 2 / True / False for rest
K	Path2D1	Size = 1 / End of program.	Size = 1 / Infinity loop



A.3 Gruppekontrakt

- Gruppe Navn: JESTA
- Vejledere: Troels Bjerre Sørensen & TA Rasmus Greve
- Kursus: Førsteårsprojekt
- Data: 28-02-2015

A.3.1 Medlemmer

Amanda

- MBIT: ENFJ (Teacher)

Emil

- MBIT: ESTJ (Supervisor)

Jakob

- MBIT: INTJ (Mastermind)

Stefan

- MBIT: ENTJ (Field Marchal)

Thor

- MBIT: INFP (Healer)
- Kontaktperson

A.3.2 Retningslinjer

Kommunikation

- Primær kommunikationsmiddel er Facebook
 - Alle er på Facebook
 - Vigtige beskeder (fx. vedr. travær) skal kommunikeres i en separat besked på vores fælles Facebook-gruppe eller alternativt via SMS.
- Intern kommunikation skal være klar og entydig
 - Således minimeres antallet af misforståelser
- Hvert gruppemedlem har ansvar for at holde sig ajour med informationer i gruppen

Værktøjer

- Google Drive
 - Bruges til skriftligt arbejde relateret til rapporten under arbejdsprocessen
 - Det vil sige vi arbejder med Google Documents, når vi arbejder på et fælles skriftligt arbejde under møder
- Latex
 - Alt skriftligt arbejde inkorporeret i rapporten afleveres endeligt i Latex
- Feedback session (inspireret af LP - en ordstyrer, en person gennemgår udfordring og reflektions-team observerer og giver feedback)
- Versionsstyring via Git
 - Muliggør samtidig arbejde og undgår at overskrive data
 - Registrering fortæller, hvem der har ændret hvad og hvornår
 - Man undgår konflikter og kan arbejde på nye funktionaliteter uden at påvirke hovedprogrammet via branches
 - Det er muligt at fortryde ændringer
 - Giver et overblik over arbejdsprocessen i softwareprojektet

A.3.3 Principper & arbejdssnормер**Aftaler skal overholdes**

Der er mødepligt ved alle faste gruppemøder, ikke løse mødedage som er angivet i mødeskemaet. Man giver besked, hvis ikke man kan overholde aftalerne, så man ikke spilder hinandens tid. I tilfælde af at en person ”lovligt” er fraværende må det forsøges på bedste vis at inddrage denne person i beslutningen. I tilfælde af sygdom eller for hindring af fremmøde er man selvfølgeligt undtaget normale møderegler.

Gensidig respekt

Selvom man er uenig, er stresset osv. skal man respektere hinanden og imødekomme alle in- put. Alle gruppemedlemmers taleret skal respekteres, og vi afbryder ikke hinanden.

Konstruktiv kritik

Man kritiserer ikke for at kritisere, men kommer med forslag til hvordan tingene kan gøres bedre. Ligeledes skal man også kunne klare konstruktiv kritik uden at tage det personligt.

Korrektur

Man må meget gerne rette fejl i hinandens formuleringer til det bedre – hvis man er uenig omkring en specifik formulering, skal det tages op til diskussion i gruppen.

Fælles ansvar

Alle gruppemedlemmer har et fælles ansvar for at nå deadlines og den generelle dagsorden, og det er vigtigt, at alle gør sit. Vi arbejder alle seriøst og fokuseret, den ugentlige aflevering indbefattes af det fælles ansvar.

Arbejdsfordeling

Der er ikke nogen i gruppen, der skal overbebyrdes med arbejde. Arbejdet skal uddeles retfærdigt mellem gruppemedlemmerne.

Konfliktløsning

Intern snak i gruppen om problemet. Evt. anonym skrive runde. Ordstyrer i gruppen med enkelt medlem der taler og resten af gruppen giver feedback efter gruppe medlemmet har talt ud.

Beslutninger

Alle vigtige beslutninger vedtages fælles ved gruppemøderne men i tilfælde af, at man ikke kan nå til enighed, bestemmer flertallet. Alle gruppemedlemmer har vetoret, men man skal kunne give en acceptabel begrundelse for, at veto benyttes.

Ærlighed/åbenhed

Hvis man er utilfreds med noget i gruppearbejdet, skal man bringe det op til diskussion i gruppen og ikke fortie det.

Mål

Gruppen skal være enig om det fælles mål for opgaven – dvs. hvad vores opgave skal under søge og uddybe.

Spørgsmål

Der findes ingen dumme spørgsmål. Man skal ikke være bleg for at spørge sine gruppemedlemmer, hvis der er noget, man ikke forstår.

Kompetencer

Hvis der er et gruppemedlem, der har en speciel kompetence indenfor f.eks. programmering, er det oplagt at bruge vedkommendes kompetence. Man skal således anerkende og udnytte hin andens kompetencer, dog lægges der også vægt op individuelle medlemmers ønske om at arbejde med noget ukendt for at lærer.

Ambitionsniveau

Gruppen har fastlagt et højt ambitionsniveau. Dette indebærer, at vi alle stræber efter et 10-tal eller bedre og arbejder målrettet og fokuseret på at opfylde dette.

Sanktioner

Sanktioner er delt op i tre “grader”.

- 1. Grad - giv kaffe til gruppen
- 2. Grad - giv sødt til næste møde.
- 3. Grad - giv øl i fredagsbar

Graderne pålægges alt efter om man har tilegnet sig en sanktion tidligere samme uge. Kommer man fx senere end de 15 akademiske minutter inden for mødetiden uden melding dagen forinden pålægges man en sanktion.

Gruppemøder

Gruppemøder foregår på ITU medmindre andet er aftalt. Hygge aften foregår primært hos Stefan.

A.3.4 Mødestruktur

Gruppen er alle enige i, at en mødestruktur er essentiel for det kommende gruppearbejde. Mødestrukturen er opdelt som følger:

Starten af møde

- 15 akademiske minutter.
- Fastsættelse af dagsorden
- Udvælgelse af referent og ordstyrer
- Klarlæg arbejdsopgaver

Under møde

- Arbejde med dagens arbejdsopgaver.
- Referent tager noter til mødet.
- Referent observere og notere tidsforbruget til arbejdsrelateret aktivitet.

Slutning af møde

- Evaluering og opsummering af dagens arbejde.
- Referat.
- Eventuelle hjemmeopgaver.
- Klarlægning af næste dagsorden.

	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag
8-10	x						
10-12	x	x	~			~	
12-14	x	x	~			~	
14-16	x	x	~		x	~	
16-18		x			½		
18-20		x					

x → Fast mødepas.

½ → Halvt møde pas (1 time).

~ → Valgfri mødetid / skal aftale med gruppen efter behov.

A.3.5 Mødetidspunkter

- Bruger ca. 30 min på planlægning i starten af hvert møde.
- Bruger ca. 30 min på oppsummering ved 15-17 tiden, når folk er ved at blive trætte.
- Skriver på rapport løbende og bruger 1-2 timer ugentligt på “næste skridt” om fredagen

Gruppearbejdet skal så vidt muligt planlægges i undervisningstiden, dvs. fra 8.00 - 16.00. Lørdag bruges ved nødsituationer, dvs. den kun tages i brug ved eksamenssituationer, eller hvis gruppen har svært ved at nå en deadline.

Pauser

Pauser skal begrænses til 5-15 minutter og frokostpausen afholdes frokosttid. Alle gruppemedlemmer har ret til én pause én gang i timen, hvis man ønsker noget at drikke eller at gå på toilettet. Hvis et gruppemedlem bliver dårlig, er vedkommende selvfølgelig undtaget fra denne regel. Generelt holdes pauser kollektivt.

A.3.6 Koderegler

Code Review

- Opret ny branch med passende navnekonvention (feature, bug eller tredje, fx. Feature/zoom_out_function)
- Dokumentation, kommentarer og evt. Testing
- Kodegruppens arbejde bliver godkendt af tredje gruppemedlem
- Pull ændringer fra master og push derefter egne ændringer til master, hvis forgående gik vellykket
- Slet branch

OBS

- Folk får uddelt kodeopgaver i grupper gennem kollektiv enighed i hele gruppen
- Ingen laver noget på egen hånd uden at have aftalt det med resten af gruppen først.
- Folk skiftes til at kode og skrive på rapporten.

Commit

Ved commit skal der i den tilknyttede kommentar udspecifceres, hvad der er lavet; eksempelvis om noget er blevet tilføjet, ændret eller slettet (added, modified, deleted).

A.4 Logbog

A.4.1 25-02-15

- Ordstyre: Thor
- Referant: Emil

Dagsorden

- JavaFX vs Swing - 20 min
 - Diskussion af GUI
 - Start med swing til lørdags aflevering og derefter søge råd hos TA's om hvorvidt vi bør holde os til Swing.
- Spørgsmål til TA's
 - Inner-Outer patterns, how-to?
 - Optimal strukturering af kode (model) ?
- Gruppekontrakt / Forfatning

JavaFX .

Fordele:

- Det nye, fedt at lærer det at kende
- Meget flottere og flere valg
- Fremtidig brug på arbejdsmarkedet

Ulemper:

- Nyt, skal læres fra bunden
- Manglende dokumentation og how-to's
- Bruger meget plads og tid (bufferedImage)

Swing .

Fordeler:

- Effektivt ift hvad programmet skal kunne
- Mange års dokumentation
- Nemt at gå til

Ulemper:

- Swing på vej ud
- Ikke nær så flot som JavaFX

Fællesmøde: 1 time. Afslutende møde: 30 min.

Dagens opgaver:

- Github + branches
- Gruppe kontrakt
- Latex
- Code Review

Til næste gang (lørdag d. 25/3)

- Færdig gruppeforfatning, renskrevet i Latex
- Oversigt over tidsforbrug
- Oversigt over nuværende features og funktionaliteter i version 1
- Oversigt over ønskede funktionaliteter til fremtiden i version 1
- Screenshot af version 1
- Dokumentation og kommentarer af kode

A.4.2 28/02-2015**Dagsorden**

- Dokumenter og kommenter kode.
- Gruppeforfatning færdig i Latex
- Oversigt over tidsforbrug
- Oversigt over nuværende features og funktionaliteter i v0.1
- Oversigt over ønskede funktionaliteter til fremtiden i v0.1
- Screenshot

Dagen opgaver:

- Dok. og kom. kode
- Oversigt over features, nutidige og fremtidige
- Skrive gruppeforfatning rent i
- Latex
- Implementering af zoom og drag
- Restrukturering af model og view + GUI
- Mockup af evt. Gui
- Blackbox testing

A.4.3 2/3 - 2015

Snak og udførelse omkring omstrukturering af klasserne

- Rydde op i klasserne.
- Klasse design og omstrukturering af disse. → Skaber klarere arbejdsfordeling.
- Laver holdopdeling og arbejder på dette.

Swing og JavaFX

- Casvas laves i swing → Overføres til JavaFX. Dvs. Canvases / selve kortet laves i swing, og resten af GUI'en laves i JavaFX. Evt. ved brug af Buffered Image.

Diverse

- No bugs at this point.
- Make tests! → Men oprydning først.
- Skal ændre windows size til 2 int variabler i stedet for at hardcode.
- Inner og outer?

Plan for idag

- Oprydning af klasserne. Start 10.15

Dagens opgaver:

- Gennemgang af kode dokumentation
- Ordning af Git
- Oprydning af kode
- Tests
- Planlægning af næste dagsorden / wrap up på i dag

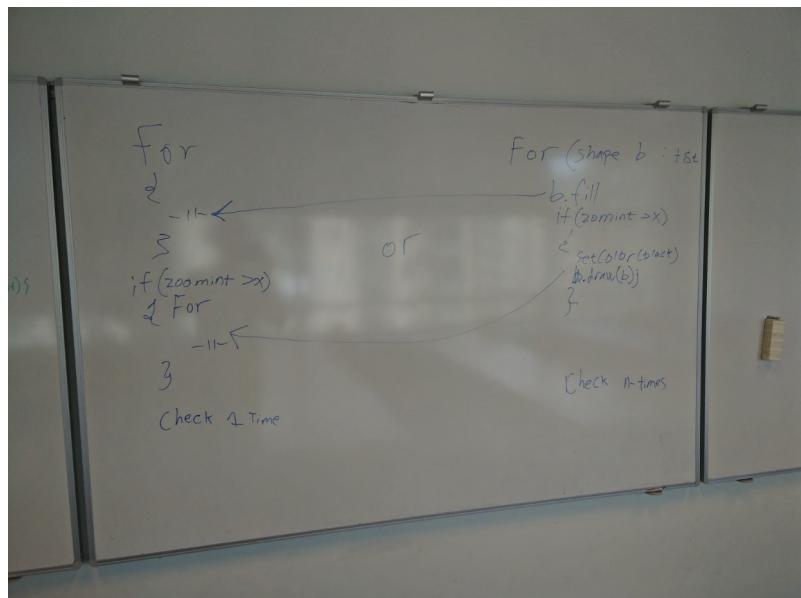
Plan for imorgen

- Fortsat arbejde med tests
- Designbeslutninger! (klasse hierarki) / planlægge GUI
- Arbejde på kun at tegne indenfor vinduet (få den til at kører mere smooth.)

A.4.4 3/3 - 2015

Dagsorden

- Designbeslutninger (klasse hierarki)
- Implementation af Draw class / shape class skal holde subklasser (fx. building)
- En forløkke -> mindre kontrol
- Kontrol (if else) koster tid i for løkker
- Kørertid, forskellige loops alt efter if
- Subklasser - større sammenhæng i vores program
- evt. abstrakt klasse MapObjects



Planlægge GUI
Arbejde med tests
Få programmet til at kører mere smooth, ved kun at tegne
inden for programvinduet
Begynde på Rapport!

Dagens opgaver:

- Klassehierarki
- Rapportskrivning

- Implementation af drawShape
- Testing af controller

drawShape implementation

- Oprette instanser af subklasserne i DataCollector
- Flytte arraylister fra ShapeCollection til subklasser
- Skriv SubClass.getList istedet for ShapeCollection.getClassList;
- Længere sigt: opdeling af DatabaseCollector med ny klasse som gemmer data, så håndtering kun sker i Collector og data gemmes andet sted.
- Map Paint kommentar:
 - Vi bruger to for løkker til at tegne indhold og streger omkring istedet for 1, da det giver mere kontrol når man fx. vil udelade noget afhængig af zoomniveau.

9/3 - 2015

- Ordstyrer: Thor
- Referent: Emil

Dagsorden

- Ændring af mødetidspunkter
- Fredag skal være fast dag 15-18 til rapportskrivning
- Lørdag bruges på hjemmeopgaver i undergrupper

Nye arbejdsopgaver i undergrupper:

- Thor og Jakob arbejder i feature/Controller_parser:
 - Address search & parser
 - Tegn kun indenfor vinduet
 - HashMap istedet for arraylist (optional)
 - Constant istedet for amortized constant time
- Amanda, Emil og Stefan arbejder i branchen feature/GUI_dynamicResizeCanvas
 - Dynamisk resize af canvas
 - * Canvas skal justeres når man hiver i vindet
 - * Knapper og søgefunktioner skal forblive samme størrelse
 - Vindue og knapper i GUI
 - * Load knap

- * Søgefelt som tager en adresse
 - Tegn punkt ved adresser som er ens
 - Adresse består som start af vejnavn, nummer og postnummer/by
 - Fra A til B felter
 - Genkendt rute tegnes op med mørkeblå
- * Save knap til ruter eller til at gemme kortet
 - Så det ikke skal loades ind hver gang man fx. zoomer

Testing skal færdiggøres

- Controller
 - keyPressed - Mockito, invokeLater - Emil og kompagni
 - Opret ny MVC, x skal være View scale
 - parser input - Thor og Jakob

Dagens opgaver:

- Clean Code workshop
- Feedback
- Separation of inner classes
- Fjern magiske tal

Tid

- Mandag: 6 timer (9-16) minus 1 frokosttime)
- Tirsdag: 7 timer (10-18 minus 1 frokosttime)
- Fredag: 3 timer (15-18 uden pauser)
- Lørdag: 4 timer (11-16 minus 1 frokosttime)
- Sammenlagt 20 * 5 timer

10/3 - 2015

Dagens opgaver:

- Refaktorering af indre klasser
- Gennemgang af feedback fra Code Review
- Dokumentation og kommentarer
- Street names
- Rapport småting
- Test af street names

Ny designbeslutning

- Canvas er blevet rykket ud af View.Map, OSMHandler er blevet rykket ud af Model.DataCollector og FileMenu er blevet rykket ud fra View.Map.
 - Formål: mere modulært design med bedre overblik over ansvarsområder
- Informationer lagres i HashMaps istedet for arrays pga konstant tid (performance issue)

16/3 - 2015

Dagsorden:

- Thor, Stefan - Rapport
- Amanda, Jakob - Street names
- Emil - GUI, brugerinterface
- Opsamling
- Afslutning af møde

Dagens opgaver:

- Rapport
- Street names mac version
- Git problemer
- GUI
- Fælles orientering / opsamling
- Rapport billeder

Punkter til næste møde

- Forklaring af gridbaglayout
- gridbagconstraints
- panel

17/3 - 2015

Arbejdsopgaver:

- Amanda → arbejder på at addresser skal kunne ses på mac også (virker på windows).
- Jakob → arbejder på at kunne vise ikoner på kortet.
- Carlos → arbejder på at få search knappen i GUI til at virke og at dokumentere noget af hans kode, og skriver Rapport.

- Stefan → arbejder på at få address parseren til at tage i mod en adresse kun med vejnavn, hjælper Carlos med GUI, og skriver rapport.

Status:

- Amanda's task er complete, men skal justeres.
- Jakob's task er complete, men skal justeres.
- Carlos' search knap virker endnu ikke, men han arbejder nu på rapport.
- Stefan's address parser kan tage imod en adresse kun med et vejnavn, og arbejder på rapport nu.

Tid brugt: 8 timer x 5 = 40 timer

20/3 - 2015

- Taleemner:
 - Rapport opg. uddeling
 - Weekendarbejde hvor og hvornår?
- Stefan - datastrukturer
- Jakob - Canvas og drawShape
- Thor - Kodeprincipper
- Amanda - implementation af vejnavne

Ting i rapport Analyse

- Implementation af brugergrænseflader - Emil og Jakob
- Algoritmer - Amanda
- Datastrukturer - Stefan

Teknisk beskrivelse

- Struktur - UML diagram - Amanda

Testing

- Blackbox og whitebox
- Test af parser, Thor

Brugerguide Mangler, videreudvikling og konklusion Refleksion over projektet

Søndag Rapportskrivning

23/3 - 2015

Forelæsning + fremlæggelse = 4 x 5 timer = 20 timer Møde = 30 min x 5 = 22.5 timer

24/3 - 2015

Dagsorden:

- Forklaring af GUI panel og komponenter af Emil
- Plan for dagens arbejde
- Planlægning af dato for social arrangement efter påskeferien

Dagens opgaver:

- Afsnit om funktionelle tests og UI tests
- Dokumentation af GUI i BuildAllComponents
- Inner Outer
- Implementation af vejnavne
- Datastrukturer i OSMHandler (QuadTree)
- Relations i OSMHandler
- Address Parser
- Kodeeksempler og rettelser i rapport
- Afsnit om projektværktøjer
- Latex Import Pygments fiks
- Nye shapes (øer, badebroer m.m.)
- Skriv om drawShapes og subklasser
- Code Review til Canvas og OSMHandler inner outer

Problemer

- Vores address parser godtager ikke addresser bestående af flere ord. Desuden kan en hurtig løsning medføre, at den ikke kan skelne mellem adresse og by, fx. ved at skrive Rued København.
- Vi har implementeret en ny faktor for zoomniveauet kaldet 'truezoomlevel' som ikke er hard coded. Vi vil dog gerne forstå dens størrelsesorden, så vi har en idé om hvor langt inde på kortet vi er. Dette er for at imødekommet kravet om geometriske afstande og informationer. Det kunne eventuelt fremstå på to labels tilknyttet hvert koordinat i hjørnet af vores FrameView.
- Vejnavne skal helst stå sideordnet med veje og ikke i slutningen af vejen.

- Listeduplikation. Der er dobbelt så mange referencer og Path2D objekter i listerne for ShapeCollection. Kunne ændres til en databank

Fixet

- Inner Outer er fikset for bygninger via Canvas og OSMHandler ændring
- Fremtid: Path2d istedet for Shape
- Canvasændring til Inner Outer

Forklar setWindingRule funktion: En bygning kan have to indre gårde. 1) Starter udenfor bygning 2) Går ind i polygon og finder polygoner 3) Winding Odd finder kun de ulige polygoner, dvs. alt omkring de indre polygoner. (den ydre polygon)

Dagens opgaver

- Afsnit om funktionelle tests og UI tests
- Zoom Level størrelsesorden
- Datastrukturer i OSMHandler (QuadTree)
- Logbog
- Afsnit om projektværktøjer
- Latex Import Pygments fiks
- Code Review til Canvas og OSMHandler inner outer
- Undervisning
- Præsentation
- Code Review til refaktorering fra Shapes til Path 2D
- Spørgsmål til Troels

30/3 - 2015

Dagsorden: Projektmøde og feedback med Troels Koordinering af rapportskrivning i påskeferien

Resultater: 1) Udjævning af git aktivitet + backlog 2) Søndag er fremover deadline dag for rapport, hvor Amanda samler ting i latex, læser korrektur og indsætter kodeeksempler, hvor folk ønsker det. 3) Efter ferien afholdes code session om datastrukturer 4) Thor undersøger implementationen af et quadtree og laver en skitse

Rapportskrivning i påskeferien: Følgende emner er blevet fordelt gennem enighed blandt de 4 tilstedeværende gruppemedlemmer fra dagens projekt- og gruppemøde. 1) Thor skriver om datastrukturer (quadtree) og laver backlog 2) Amanda skriver om streetnames under gui, hvordan man kan gøre det ellers og indsætter eksempler til eksisterende afsnit i rapport 3) Jakob skriver om Canvas, drawShapes, shapecollection og inner/outer relations 4) Emil skriver om gui 'før/nu', brugerguide og

ddyber evt javafx vs swing afsnit 5) Stefan skriver om implementationen af address parser og afsnit med UML (klassediagrammer)

Implementation i påskeferien: 1) Thor udarbejder skitse til QuadTree 2) Jakob og Emil fikser indsættelse af nyt punkt til udregning af vinkler og rotation 3) Jakob kigger efter påskeferien på issue med dobbelt liste (dobbelt hukommelse) i relations

Hjemmeuge 6/4 - 12/4

Opgaver:

- Backlog
- BufferedImage load funktion research
- RangeSearch research
- AffineTransform research
- Afsnit om RangeSearch
- Afsnit om AffineTransform
- QuadTree implementation
- BufferedImage loading
- StrokeSetter Tag
- QuadTree Implementation
- BufferedImage Loading
- Save i binær format

Stor designbeslutning: separeret data i subklasser skal samles ét sted

QuadTree issue Problem: QuadTree kan ikke skelne mellem forskellige bygninger med nuværende kodestruktur Løsning: sender Path2D-objekter til QuadTree istedet for Shape Lister Undgår extra memory på HashMaps

Problem: Opdeler ikke længere, Canvas som får informationer fra QuadTree ved ikke hvad de forskellige Path2D objekter repræsenterer Løsning: Lavet en Path2DDra klasse med et unikt tag StrokeSetter håndterer tags så tingene på kortet bliver tegnet ordentligt Fylder 4 bytes ekstra per Path

Fordel: I ShapeCollection er colorMap(Path2D, int) blevet fjernet: for hver streg lavede vi før ekstra HashMap, men nu er den fjernet og har halveret memory space

Sammenlagt: nu er alt information samlet i quadtræet istedet for at være spredt omkring subklasserne

Midlertidig løsning: HashMap som erstatning for quadtree

Performance issue Hvad værdi giver det at tegne fx. bygninger, når man vil se korteste rute fra A til B? Ved loading af kort skal der kun være veje som udgangspunkt, slår selv bygninger til ViewBox grænserne der bestemmer kortets grænser i programvisningen skal bruge et BufferedImage til at loade kortet istedet for at gentegne heletiden

Vigtig designbeslutning

DrawShape og subklasser for shapes bliver fjernet fordi: DrawShape holder på information omkring tegneprocessen (View opgave) og de rå data (HashMap for hver subklasse med rådata) som modstrider MVC Gråzone med designstruktur DrawShape kender til hvordan subklasserne tegnes men det bør egentlig afgrænses til subklasserne alene QuadTree issue Kræver alle informationer fra ét sted til opdeling af kortet Laver istedet subklasser for tegneprocessen hos StrokeSetter

Save OSM i binært format understøttes nu ved at Path2DDraw klassen implementerer Serializable Mål StrokeSetter tag Erstatter shape subklasser med unikke tags brugt i StrokeSetter (vigtig beslutning) Fjerner HashMap i Fields Udfordringer ZoomNiveau ved ikke om tingene skal tegnes da det hele er samlet i en liste → Zoom niveauer skal afgøres af float tag Parker risikerer at overtegne søer, fordi rækkefølgen hvorpå ting tegnes på kortet er tilfældig Kunne lave prioriteringskø eller midlertidig liste

Uge 15

Opgaver:

- StrokeSetter Tag
- Save/Load function
- Afsnit om Range Search
- Afsnit om AffineTransform
- Changelog og backlog
- Refaktorering af HashMaps + ny PathTag klasse
- Flere Relations
- Fulde addresser i “addrMap”

13/4 - 2015

Dagsorden:

- Gruppepræsentation
- Koordination af ugens målsætninger
- Ny datastruktur
- BufferedImage load / save
- Rapportskrivning

Dagens opgaver

- Forelæsning
- Gruppepræsentation
- Implementation af BufferedImage load
- Autopan og vejnavne
- Refaktorering (Canvas, PathTag og StrokeSetter)

14/4 - 2015

Dagens opgaver:

- at loade Buffered Image
- Auto Pan og adressesøgning
- boundaries (kommuner)

Dagens opgaver

- Refaktorering
- Klassegennemgang
- BufferedImage koordinater
- AutoPan
- Relations

Klassegennemgang

DataCollector Får en fil og sætter OSMHandler eller TXT-Handler til at håndtere den. “Hvilken fil er det og hvordan skal den behandles” Refaktorering: isInDrag, setInDrag og get/set X skal refaktoreres til MouseController

OSMHandler Modtager OSM-fil fra DataCollector som den henter relevante informationer fra via tags. Informationer gemmes i HashMaps i “ShapeCollection”. AddressMap<String, Long> holder på en adresse til reference. ReferenceMap<Long, Point2D> holder på en reference til et punkt. Path2DDrawMap<Long, PathTag> holder på en reference til et pathTag/vej/bygning. Refaktorering: Long til Float (backlog) startElement: Hvis starttag er Node finder den ID som bliver til reference Lat og Lon bliver til Point2D PathTag (Path2D) bliver skabt når to Point2D eksisterer Tags beskriver Nodes og referencer til nodes, fx. addr: street (gadenavnet) Giver PathTag objekter en int der afgør design -> beskriver udseende I starten af hver eneste OSM fil er der en lang liste af Nodes Nd er en reference til allerede eksisterende nodes som bliver forbundet sammen **Relations** Refererer til Nd og Node Kigger på deres indhold, Member tags som holder på ND og RoleÆ Member: “Nd” og en “Role”, fx. outer role Hvis du har en outer og en inner så bindes de sammen i vores program Hvis vi havde en bygning med gård ville den blive farvet med samme farve før Med inner/outer kan vi appende (tilføje) gården (ideal) Relations

er ikke færdigudviklet, fx. hvis der kun er outers og ingen inner, hvordan bindes de, eller hvis de er tomme? To/from (ensrettet)? Boundaries for kommuner her

Veje Veje er bygget op af små streger Eksempel: Hvej Brudstykker Hvej1, Hvej2, Hvej3.... Sidste vej overskriver forgående Ved de sidste 2 punkter: fx. Hvej N-1 og Hvej N Idé: Får vejnavn skrevet ind i slutningen Vi tegner indtil de 2 sidste

ShapeCollection Holder på alt rå data fra OSMHandler eller FileHandler/TXTHandler Refaktorering: Når View klassen skal hente info herfra skal den altid gå igennem DataCollector som har ShapeCollection -> skal hente information direkte View og Canvas skal modtage ShapeCollection (i field) direkte

PathTag Vores udgave af Path2D.Double Holder på et int tag der beskriver hvad for en Path der er tale om (bygning etc), en slags designtag brugt i StrokeSetter og Canvas til at tegne objekterne på kortet rigtigt.

AddressParser Modtager en adresse fra søgerfeltet som den sammenkobler med nogle regulære udtryk for hvordan en korrekt adresse ser ud. Adressen kan bruges til at forsimple adressen, fx. "mente du dette"? Videreudvikling: nested char array

Calculator Modtager 4 punkter (minlat, minlon, maxlat, maxlon). Returnerer double værdi som måler afstand fra et punkt til et andet. Den tager højde for jordens krumning. Bliver brugt i viewBox til at bestemme hvor vi er på kortet. Værdien den returnerer har en bestemt størrelsesorden som bør laves om til meter værdi. Skal bruges til afstandsmåling som skal bruges til shortest route. Refaktorering Metoder til loading af BufferedImage i Canvas skal rykkes ud i ImageLoader i View FileChooser skal sammenkobles med FileLoader og FileSaver i Model Addressefeltet i programmet skal have sin egen controller klasse "Textfield" i

Controller package Gennemgang af View og bbox (viewBox) og deres værdier

MapView samler det hele Overordnet builder klasse bygger subelementer Panel laver JPanel og har GridBackLayout som alle byggeklasserne har GridBackLayout er anderledes end GridLayout (bruger vægt til forskellige ting) JFrame holder BorderLayout (NSWE) Canvas er center i BorderLayout Panel er east men kan ændres til west Panel indeholder GridBackLayout GridBackLayout ændrer sig ift. komponenternes størrelse Hard coded størrelser på panel, fordi størrelsen ellers ville ændre sig når man skriver noget ind i tekstfelt ComponentBuilder initialiserer alle delfunktionaliteterne og holder på referencer til dem Fields fungerer som referencer

Canvas Menubar laver JMenuItem i toppen (ingen funktionalitet endnu) FileChooser implementation iMenuBar ComponentBuilder refererer til subklasser Problem: når vi panner bruger vi pixelværdier istedet for længde og breddegrader viewBox issue... Konvertering af pixel til latitude og longitude

BufferedImage issue BufferedImage skal vide hvad den skal tegne Quadtræet får måske 1/10 som tegnes, får et BufferedImage og bliver sat ind Behøves ikke repainte når man panner og zoomer

bbox er statisk (vores udgangspunkt zoomet helt ud) og viewBox er det vi ser dynamisk viewBox skal være større end det på skærmen, fx. faktor 2

DrawnCorner = 0-canvas.width/2; LeftUpperCorner =getPointOnMap(drawnCorner, drawnCorner) Antager at vindue er lige bredt og højt

DrawnCorner = canvas.getWidth+canvas.width/2; RightLowerrCorner =getPointOnMap(drawnCorner, drawnCorner)

Mål: centrer rektangel på midten af, det vi tegner

18/4 - 2015

Hjemmearbejde

- Serialization
- FileChooser

19/4 - 2015

Hjemmearbejde

- Binary Load Function
- FileChooser

20/4 - 2015

Dagsorden

- Implementerer setter til QuadTree i ShapeCollection, som skal bruges til at loade deserialized objektdata ind fra quadtree gemt i binær format
- Færdiggør BufferedImage loading
- Diskussion af ruteplanlægningsklasse og API
- Færdiggør relations
- Påbegynder ruteplanlægning efter diskussion

Mål: Indledningsvis at kunne finde den korteste vej mellem to punkter på kortet. Dernæst at kunne finde den hurtigste vej mellem to punkter baseret på vejtyper og hastighedsgrænser. Til slut at kunne bruge en heuristik, der gætter distancen ud fra en vis vægt forbundet til ruten. Heuristik gør vægte mindre i den rigtige retning, så søgningen indsnævres til et mindre område indeholdende destinationspunktet.

API: Relax edges (se bort fra kanter efter brug) Pålægge Nodes en prioritet ud fra estimat fra heuristik Udregne afstand mellem to punkter (distTo) hasPath (eksisterer der overhovedet en sti mellem punkterne) Check metode til at se om heuristik er korrekt Den estimerede afstand må ikke være større end den reelle afstand (admissible) Den estimerede afstand for et hvilket som helst nabopunkt ved siden af destinationspunktet må højest være det samme som nabouheuristikken + prisen for at komme derover Heuristikken for naboen må højest være det samme som heuristikken for et nabopunkt + afstanden mellem start og nabopunkt

Implementation:

Udfordringer: Normal Djikstra algoritme finder korteste rute for alle distancer, der tilsvarer distancen fra a til b (hele cirklen). Djikstra fra bogen stopper ikke med at lede så snart destinationspunktet er fundet. Find en algoritme på Princeton, der implementerer en heuristik og hvis det ikke findes, så undersøg hvordan den implementeres.

Designbeslutning omhandlende tegning af Relations

Nuværende situation: Udviklingen fra Hashmaps til QuadTree har fjernet de referencer til PathTag objekter, som vi bruger til at finde Relations med. To løsningsmuligheder: Programmet kan først læse Relations og holde på de relationer som skal bruges Vi genskaber ét HashMap som holder på referencer og PathTag objekter. Når relationerne er lavet (bundet op af flere veje fra før) overføres dataen til quadtræet, imens HashMappet gøres mindre Når vi er noget til enden af OSM-filen skal alt fra HashMap overføres til QuadTree, og undervejs processen skal HashMappet mindskes automatisk

Test: måler med en Profiler hukommelsesforbug 3 gange for hver løsning (nuværende og ny kodestruktur)

Planer for imorgen d. 21/4 Fiks loading a binær fil Afprøv programmet på hele Dan markskortet
 Påbegynd ruteplanlægning Track hukommelsesforbrug med VisualVM Gennemgå rapportskrivning
 Ændring af variabler initialiseret i metoder og kørt mange gange (fx. objekter oprettet i startElement()
 i OSMHandler der bliver oprettet 45.000 gange hver især for kortudsnit af København)

Designbemærkning: I OSMHandler initialiserer vi objekter indeni metoder istedet for at overskrive dem som felter, for Bornholmkortet initialiserer vi de 4 objekter 230.000 gange hver især

Dagens opgaver:

- Modeldiskussion
- Ruteklasse planlægning
- Relations designbeslutning
- Autopanning
- Binary Load Function
- FileChooser
- Relations
- Shortest Path research

Ruteklasse planlægning Dagens opgaver

- Suggestionbox
- Koordinering og mål
- Binær load
- Ruteplanlægning
- Range Search
- Canvas Refactor
- Git Merge conflict
- BufferedImage loading

- Binær load CPU performance issue (paint og load)
- Branch merge conflicts

QuadTree udfordring på Bornholmudsnit: QuadTree afgrænset ikke kun det viste kort og tegner derfor ting omkring også Rektangel omkring Bornholm er stor men indeholder ikke mange informationer Serializable påvirker memory consumption?

Binær load Compilation af Danmarkskortet tager 6 min uden binær format Compilation af Danmarkskortet tager med binær format et par sekunder

CPU: paint og load

Spørgsmål til Troels CPU: paint og load i Canvas, hvordan optimerer vi? Serializable andet vi skal gøre?

Autocomplete til adresser virker Datacollector: får liste fra shapeCollection (getAddressMap) HashMap laves om til objekt array (JComboBox auto completion tage denne type liste) -> kunne også være string array MapView tager DataCollector ved initialisering i main Kan ikke lave ny DC, fordi den ikke har liste fra OSM fil MapView getter liste og smider i konstruktor for panel, der nu tager en liste af Objects Panel extender ComponentsBuilder Fra Panel kan vi getJComboBoxList som også tager Objekt liste, metode ligger i ComponentsBuilder men kan kaldes fordi Panel nedarver det Ekstern library fra Maven gør det muligt at auto complete i getJComboBoxList Java Swing Components kan ikke køre på samme thread som resten af programmet, så vi har lavet en ny runnable i main Object[] gør det muligt at tage postnumre også, ellers kan String array også bruges

26/4 - 2015

Dagens opgaver:

- Modeldiskussion
- Ruteklasse planlægning
- Relations designbeslutning
- Binary Load Function
- Gruppemøde
- Koordinering og mål
- Canvas Refactor
- Branch merge conflicts
- FileChooser
- Trello oprydning

27/4 - 2015

Dagens opgaver:

- FileChooser
- Danmarkskortet gennemgang
- Shortest Path gennemgang
- Trello og krav gennemgang
- FileChooser

28/4 - 2015

Dagens opgaver:

- Danmarkskortet gennemgang
- Shortest Path gennemgang
- Trello og krav gennemgang
- FileChooser

Status:

- Adresseændringer for mindre hukommelsesforbrug
- Adresser fylder meget
- Fjerner delinformationer
- Post -> city er unik
- 30% sparet
- Antager at postnummer er unik
- char[] istedet for String[] for mindre overhead
- Relations disabled -> ser pænere ud
- Ny HashMap implementeret

Directed Edges oprettes i OSMHandler Way PathTag holder på to punkter (vertices) Når vi læser har vi en counter til vertices, laver directed edges mens vi læser Prioritetskø: bruges når korteste vej skal findes efter deres vægt Red black tree: 2-3 træ med pænere implementation Humlen: søger efter values (vægt) tilknyttet punkter på kortet Hashe fylder meget! (HashMaps punkt til Directed Edge) Problem: vi søger på adresser og det er vejnavne vi skal finde til (smart med visning af, hvilke adresser, der er tættest på)

Coastlines long = startpoint Flere ting der starter samme sted har kun ét begyndelsespunkt (cost efficient) long = endpoint nd: reference til point2d way = pathTag (CLP, coastline path) Tager

start og slutpunkt, tjekker om der er CLP hvis endepunkt er lig startpunkt for en anden uskabt CLP (start og slutpunkt kun) Minimerer antallet af unødvenige ways som oprettes Vi antager at der ikke er huller i Relations?

Dagsorden

- KD-træ implementation til range search, hvor vi fra mus skal finde nærmeste adresse (hovedkrav)
- Point2D som key til shortest path
- Metode i OSMHandler som tager postkode og by og lægger i HashMap
- Hvis allerede eksisterer -> skip
- Tegning af DK
- Antialiasing slås fra når vi kører rundt på Danmarskortet
- Adresser udgør det største hukommelsesforbrug

3 største ting

- Float[] Alle PathTag objekter
- 48% til adresser og HashMap AdressMap (previous)
- char[] og HashNode tager 33 % ift

Designbeslutning: adresse parser vs auto completion Adress Parser vs auto completion: begge løsninger lever op til dette krav Hvis adressen er skrevet på en anden måde Adresse Parser vil kunne tage adresse på flere måder Beslutning: autocompletion istedet for adresse parser som dog medtages i rapporten

Deadline til mandag d 4 næste uge

- Ruteplanlægning
- Ny datastruktur til adresser og til at finde nærliggende adresser vha. søgeopslag i kd træ
- Binar save/load baseret på brugerbehov
- Slette MouseControllerTest da den ikke blev brugt
- Refaktorering
 - lektorReferenceMap, absoluteTree og navne på QuadTrees skal ændres i ShapeCollection
 - BufferedCanvas issue:
 - Billedet bygget op af pixels
 - Pixels indeholder int array med farvekoder (RGB)
 - Indsatte aldrig noget i dem
 - 150 millioner int arrays

l 326-343 i Canvas skal have udtrækket parametre

29/4 - 2015

Dagens opgaver:

- KD-træer til adresser
- TST (præfix søgetræ)
- Adresser er hard coded i combobox fordi GUI bliver lavet en gang (GUI er ikke dynamisk nok til at holde øje med adresser)
- Shortest Path: find adresser tættest på via sammenligninger

30/4 - 2015**Nye features**

- KD-træer til adresser
- TST (præfix søgetræ)
- Adresser er hard coded i combobox fordi GUI bliver lavet en gang (GUI er ikke dynamisk nok til at holde øje med adresser)
- Shortest Path: find adresser tættest på via sammenligninger
- Kd træer og mus til at finde nærmeste adresse -> før skulle den gennemgå 2,3 (halvdelen) af adresserne -> 5 millioner vektoroperationer
- Søg som udgangspunkt på adresse ->
- GUI (brugervenlighed, nye designvalg osv) skal skrives om i rapport
- Data Models, Task Description og Virtual Windows udgør 90% af point

Algoritmer KD træer, TST søgetrie, forbedret HashMap (adresse til punkt), punkt til adresse (KD træ) og ArrayListe (Int peger på sted i ArrayListe med adresse)

Status til deadline Path Finding rute beskrivelse (shortest and fastest path) Color Blind Mode Program skal starte på nord med kompas Dynamisk boks til visning af koordinater (hvor er man) Scalebar ændrer sig på baggrund af zoomniveau (kartografisk element)

4/5 - 2015

Dagsorden: Features skal færdiggøres Path Finding Color Blind mode Kartografiske elementer (kompas og koordinatvisning, lat = nord) Rutevejledning

MouseController: combobox settings allerede lavet

Dagens opgaver

- Dokumentation
- Refaktorering

- Shortest Path
- Rutevejledning
- Udregning af distance

Color Blind

Problem før KDtræ: dyrere at finde adresse end at tegne Danmarkskortet

5/5 - 2015

Refaktorering: Istedet for Point2D -> ADDR bruger vi Point2D til at finde vejnavne Dette er gjort som konsekvens af motorveje og små paths i fx parker, der har vejnavn men ikke adresse.

Dagens opgaver:

- Color Blind
- Merge konflikt
- Serializable afsnit

9/5 - 2015

Dagens opgaver

- Controller Package dokumentation
- PathFinding Package HTML Tag removal og dokumentation
- Model Package dokumentation
- View Package dokumentation

11/5 - 2015

Dagsorden: dokumentation, testing og evt refaktorering

Dagens opgaver

- Shortest Path Merge fiks
- Dokumentation merge
- View Package dokumentation
- MapView konstruktor refaktorering
- Address Parser i OSMHandler

Ny kodekonvention: Kommentarer til getters og setters medtages ikke såfremt de i forvejen er selvforklarende, da det virker redundant og umødvendigt.

http://www.adam-bien.com/roller/abien/entry/how_to_javadoc_efficient_and

Beslutning om sprogkompatibilitet: Programmet, dets rutevejledning og funktionaliteter skrives på engelsk

Test Test enten quadtree, kd-tree, TST eller shortest path med djikstra Vi tester ikke tri-
vielle getters/setters, da det ikke skaber særlig meget værdi og alligevel opdages løbende ved fejl, da
programmet køres regelmæssigt.

Test Package: Testklasser til DataCollectorModel, MouseController og FileChooser er blevet
oprettet

12/5 - 2015

Dagsorden: testing og refaktorering

Emil: udkommenteret kode Jakob: tester RectHV Amanda: oprydning Stefan: testklasser

Dagens opgaver

- AddressParser i OSMHandler og tests
- Oprydning
- RectHV test
- Testklasser
- Farveblindhed afsnit
- Latex
- Oprydning
- Afsnit om problemområde
- Afsnit om datastrukturer

A.5 Timelog

Uge	Fremskridt	Tidsforbrug	Pris
Uge 1 (v0.1)	Forelæsning Github + branches Dok. & Kom. kode Gruppe-forfatning Zoom & drag Restrukturering Mock-up Black-box testing	20 1 15 5 4 7 8 4	12500 625 9375 3125 2500 4375 5000 2500
i alt		64	40000
Uge 2 (v0.2)	Gennemgang af dok. Git Oprydning af kode Test Møder Canvas Design Clean code forelæsning Clean code workshop	15 3 15 10 10 28 14 10 10	9375 1875 9375 6250 6250 17500 8750
i alt:		115	71875
Uge 3 (v0.3)	Implementa- tion af Draw class Loops og køretid Sammen- hæng - subklasser MapObjects Design af GUI Klasseieraki Test Tegne inden for program- vindue Rapport Controller Møder Forelæsning	12 30 14 8 10 12 8 20 12 5 10 10	7500 18750 8750 5000 6250 7500 5000 12500 7500 3125 6250 94375
i alt		151	

Uge 4 (v0.4)	Shape-Collector	8	5000
	Refakturering	7	4375
	Data gemmes	8	5000
	Map paint	12	7500
	Møder	15	9375
	Address-parser	14	8750
	Dynamisk resize af canvas		0
	GUI	8	5000
	Genkend rute	6	3750
	Test	8	5000
	Separation af inner classes	12	7500
	Fjern magiske tal	2	1250
	Rapport	6	3750
	Forelæsning	10	6250
	Gruppe-fremleggelse	10	6250
i alt		126	78750
Uge 5 (v0.5)	Møder	20	12500
	Refakturering	6	3750
	Feedback fra Code review	8	5000
	Dok. & Kom. kode	4	2500
	HashMaps	2	1250
	Street names	12	7500
	GUI	15	9375
	Git problemer	3	1875
	Andet kode	35	21875
i alt		105	65625
Uge 6 (v0.6)	Test af parser	4	2500
	Code Review af test	6	3750
	Rapport kodenstil	3	1875
	Rapport Testing	3	1875
	Address-parser	6	3750
	Ikoner	12	7500
	Search knap	14	8750
	GUI	10	6250
	Rapport	15	9375
	Vejnavne i adresse parser	23	14375
	Møde	18	11250
i alt		114	71250

Uge 7 (v0.7)	Rapport udeling	5	3125
	Datastrukturer	9	5625
	Forelæsning	10	6250
	Gruppe-fremlæggelse	10	6250
	Canvas	10	6250
	DrawShape	6	3750
	Clean code	10	6250
	Implementa-tion af venhavne	14	8750
	Rapport brugergrænse-flader	4	2500
	Rapport Algoritmer	2	1250
	Rapport Datastrukturer	2	1250
	Rapport UML	2	1250
	Rapport test	5	3125
	Rapport Brugerguide	4	2500
	Møder	15	9375
i alt		108	67500
Uge 8 (v0.8)	Forelæsning	10	6250
	Gruppe-fremlæggelse	10	6250
	Møder	10	6250
	Plan for ugen	5	3125
	Funktionelle test	4	2500
	UI test	4	2500
	Dok af GUI	5	3125
	Inner Outer	12	7500
	Quadtree	18	11250
	Adress parser	14	8750
	Zoom level	6	3750
	Kodeeksempl er og rettelser	6	3750
	Rapport		
	Projekt værktojer rapport	2	1250
	Latex	4	2500
	Shapes	5	3125
	DrawShape rapport	4	2500
	Code Review	8	5000
	Farvebling	8	
i alt		135	84375

			v
Uge 9 (v0.9)	Inner Outer	5	3125
	Path 2D	4	2500
	Polygoner	5	3125
	Test	9	5625
	Zoom level	8	5000
	Forelæsning	10	6250
	Præsentation	10	6250
	Møde	12	7500
	Feedback	15	9375
	Rapport Datastrukturer	6	3750
	Rapport StreetNames	6	3750
	Rapport Canvas	8	5000
	Rapport inner outer	8	5000
	Rapport adresse- parser	6	3750
	Skitse Quadtree	10	6250
	Udregning vinkler og rotation	12	7500
	Relations - dobbelt hukommelse	16	10000
	KD-træ	18	11250
	Kartografiske elementer	14	8750
i alt		182	113750
Uge 10 (v0.10)	Backlog	10	6250
	Buffered- Image load	8	5000
	RangSearch	12	7500
	Affine- Transform	8	5000
	Afsnit om RangeSearch	6	3750
	Afsnit om Affine- Transform	6	3750
	QuadTree	16	10000
	StrokeSetter	8	5000
	QuadTree Implemen- tation	6	3750
	Save i Binær format	15	9375
	Møde	20	12500
	Costlines	22	13750
i alt		137	85625

Uge 11 (v0.11)	Save OSM i binær format	12	7500
	Changelog	6	3750
	Refakt- turering	8	5000
	Flede Relations	6	3750
	Fulde addresser	6	3750
	Buffered- Image load	4	2500
	Autopan og vejnavne	12	7500
	Calculator	12	7500
	Gennemgang af View	10	6250
	Serialization	12	7500
	FileChooser	5	3125
	Ruteplanlægn ing	8	5000
	Test	25	15625
	Møder	15	9375
i alt		141	88125

Uge 12 (v0.12)	Møder	12	7500
Rapport rettelser	20	12500	
Rapport Test	5	3125	
Rapport Konklusion	2	1250	
Rapport Videre udvikling	2	1250	
Rapport Reflektion	3	1875	
Rapport OSM handler	5	3125	
Rapport Carivas	3	1875	
Rapport Quadtree	4	2500	
Rapport Starter klasse	1	625	
Rapport Shortest path	5	3125	
Rapport Map view	1	625	
Rapport Rute- vejledning	1	625	
Rapport Brugerguide	1	625	
Rapport Swing vs java	2	1250	
Rapport Implementaion af brugergrænse flade	3	1875	
Rapport Problem område	3	1875	
Rapport Bilag	6	3750	
Rapport Kilder	4	2500	
Struktur	12	7500	
Nedskæring	3	1875	
Latex	5	3125	
i alt	103	64375	
Total	1481	925625 DKK	

A.6 Change-log

A.6.1 v0.1

Features for v0.1: ——————

- Ved at klikke og holde musen inde på det tegnede kort, trække derhen hvor man vil og slippe igen kan man se forskellige steder på kortet.
- Ved hjælp af musehjulet kan man zoome ind og ud på kortet, dette kan også gøres på mousepad eller ved + og - knapperne.
- Ved q og e kan man roterer kortet og man kan gøre stregerne på kortet mere simple ved x og omvendt pænere ved x igen.

Wishlist for features for v0.2: ——————

- At der kun bliver tegnet inden for det GUI som man kan se.
- En bedre GUI med mulighed for brugerinput.
- Gøre mouseDrag funktionen mere smooth (den hakker lidt nu, især med antialising på). Lave forskellige visninger af cykelstier og stier for fodgængere.

A.6.2 v0.2

Features for v0.2: ——————

- Ved at klikke og holde musen inde på det tegnede kort, trække derhen hvor man vil og slippe igen kan man se forskellige steder på kortet.
- Ved hjælp af musehjulet kan man zoome ind og ud på kortet, dette kan også gøres på mousepad eller ved + og - knapperne.
- Ved q og e kan man roterer kortet og man kan gøre stregerne på kortet mere simple ved x og omvendt pænere ved x igen.

Wishlist for features for v0.3: ——————

- At der kun bliver tegnet inden for det GUI som man kan se. - (Tæt på. Mangler finpudsning).
At bygninger bliver tegnet ordentligt ("Inner" og "Outer"-rollerne er i orden).
- En bedre GUI med mulighed for brugerinput.
- Gøre mouseDrag funktionen mere smooth (den hakker lidt nu, især med antialising på). Lave forskellige visninger af cykelstier og stier for fodgængere.

A.6.3 v0.3

Features i v0.3: _____

- Ved at klikke og holde musen inde på det tegnede kort, trække derhen hvor man vil og slippe igen kan man se forskellige steder på kortet.
- Ved hjælp af musehjulet kan man zoome ind og ud på kortet, dette kan også gøres på mousepad eller ved + og - knapperne.
- Ved q og e kan man roterer kortet og man kan gøre stregerne på kortet mere simple ved x og omvendt pænere ved x igen.
- Man kan også indtaste gadenavne, også viser programmet hvor den starter / slutter henne.

Udført for v0.3: _____

- En bedre GUI med mulighed for brugerinput. (Mangler at blive connected med "Controller-klasserne").
- Addresparseren er blevet opført.
- Der bliver kun tegnet indenfor hvad man kan se (Skal lige finpudses yderligere) Programmet kan lokalisere gadenavne og tegne en rød firkant på deres slutning / start. Forbedret struktur.

Wishlist for features for v0.4: _____

- At bygninger bliver tegnet ordentligt ("Inner" og "Outer"-rollerne er i orden). Flere test og egen OSM-fil for tests.
- Forbedre GUI'en og gøre den brugbar.
- Implementerer addresseparseren fuldt ud.

A.6.4 v0.4

Features i v0.4: _____

- Ved at klikke og holde musen inde på det tegnede kort, trække derhen hvor man vil og slippe igen kan man se forskellige steder på kortet.
- Ved hjælp af musehjulet kan man zoome ind og ud på kortet, dette kan også gøres på mousepad eller ved scrollle ind og ud.
- Man zoomer derind, hvor musen er. Dets længere man zoomer ind, desto mere vises der på skærmen. Heriblandt gadenavne og ikoner.

Udført for v0.4: _____

- Tegner nu også fæge og moterveje (Pink og røde)
- Abstrakt zoom-mekanisme, så programmet ved, uafhængigt af osm eller zip fil, hvilket "zoom niveau"brugeren er på.

- Tegner gadenavne ved slutningen af gaden. Tegner ikoner på bygningerne.

Wishlist for features for v0.5: _____

- Få knapperne til at fungerer.
- Få implementeret, at programmet kan tegne en rød firkant på deres slutning / start. Ved brug af knapper.
- Få vejene til at se ordentlige ud alt afhæning af zoom niveauet (Ordnes mandag). Vej navne bliver skrevet ordentligt.
- Hent ikonerne ind, uden at programmet lakker.
- Flere test og egen OSM-fil for tests.
- Implementerer addresseparseren fuldt ud.
- (Ny datastruktur?) - I forbindelse med Mandags fordrag.

A.6.5 v0.5

Features i v0.5: _____

- Ved at klikke og holde musen inde på det tegnede kort, trække derhen hvor man vil og slippe igen kan man se forskellige steder på kortet.
- Ved hjælp af musehjulet kan man zoome ind og ud på kortet, dette kan også gøres på mousepad eller ved scrollle ind og ud.
- Man zoomer derind, hvor musen er. Dets længere man zoomer ind, desto mere vises der på skærmen. Heriblandt gadenavne og ikoner.

Udført for v0.5: _____

- Få vejene til at se ordentlige ud alt afhæning af zoom niveauet. (Mangler finpusing)
- Hent ikonerne ind, uden at programmet lakker. (Ikonerne er nu mindre, rent pixelmæssigt, hvilket løste problemet).
- Moterveje er nu laksefarvet.
- Kan nu tegne øer, badebroer og andre små ting på kortet.
- Kan tegne ”inner”og ”outer”relations i forbindelse med nogle bygninger. (Mangler forbedringer, da ikke alle relation nødvendig fanges).

Wishlist for features for v0.6: _____

- Få knapperne til at fungerer. (Er tæt på)
- Få implementeret, at programmet kan tegne en rød firkant på deres slutning / start. Ved brug af knapper.

- Vej navne bliver skrevet ordentligt. (Er lidt pænere nu)
- Flere test og egen OSM-fil for tests.
- Implementerer addresseparseren fuldt ud. (Er blevet forbedret) Ny datastruktur: Endten k-D Trees eller Quad Trees.
- Kan tegne ”områder/ boundaries på kortet.

A.6.6 v0.6

Features i v0.6: ——————

- Ved at klikke og holde musen inde på det tegnede kort, trække derhen hvor man vil og slippe igen kan man se forskellige steder på kortet.
- Ved hjælp af musehjulet kan man zoome ind og ud på kortet, dette kan også gøres på mousepad eller ved scrollle ind og ud.
- Man zoomer derind, hvor musen er. Dets længere man zoomer ind, desto mere vises der på skærmen. Heriblandt gadnavne og iconer.

Udført for v0.6: ——————

- Hent iconerne ind, uden at programmet lagger. (Iconerne er nu mindre, rent pixelmæssigt, hvilket løste problemet).
- Moterveje er nu laksefarvet.
- Kan nu tegne sører, badebroer og andre små ting på kortet.
- Kan tegne ”inner”og ”outer”relations i forbindelse med nogle bygninger.

Wishlist for features for v0.7: ——————

- Få knapperne til at fungere.
- Få implementeret, at programmet kan tegne en rød firkant på deres slutning / start. Ved brug af knapper.
- Vej navne bliver skrevet ordentligt.
- Flere test og egen OSM-fil for tests.
- Ny datastruktur: Endten k-D Trees eller Quad Trees.
- Kan tegne ”områder/ boundaries på kortet.
- Save / load funktion.

A.6.7 v0.7

Features i v0.7: _____

- Float Tags istedet for Shape subklasser.
- Data fra Shapes subklasser er blevet samlet ét sted.
- Tegneinformationer sendes direkte fra OSMHandler til fx. Quadtræ.
- Dette gøres vha. klasen PathTag, der holder på unikt tag til alle Path typer. Det er nu muligt at implementere andre datastrukturer.

Udført for v0.7: _____

- Ikoner ligger ikke.
- Motorveje er laksefarvede.
- Program tegner sjæler, badebroer m.m.
- Inner Outer relations.

Wishlist for features for v0.8: _____

- Ny datastruktur: Endten k-D Trees eller Quad Trees.
- Save og load funktion.

A.6.8 v0.8

Features i v0.8: _____

- Ny datastruktur (QuadTree)
- Binær save funktion

Udført for v0.8: _____

- Float Tags istedet for Shape subklasser.
- Data fra Shapes subklasser er blevet samlet ét sted.
- Tegneinformationer sendes direkte fra OSMHandler til fx. Quadtræ.
- Dette gøres vha. klasen PathTag, der holder på unikt tag til alle Path typer.
- Det er nu muligt at implementere andre datastrukturer.

Wishlist for features for v0.9: _____

- Binær load funktion med tilhørende FileChooser
- Færdiggørelse af BufferedImage Loading funktionalitet

A.6.9 v0.9

Features i v0.9: _____

- Ved at klikke og holde musen inde på det tegnede kort, trække derhen hvor man vil og slippe igen kan man se forskellige steder på kortet.
- Ved hjælp af musehjulet kan man zoome ind og ud på kortet, dette kan også gøres på mousepad eller ved scrollle ind og ud.
- Man zoomer derind, hvor musen er. Dets længere man zoomer ind, desto mere vises der på skærmen.
- Man kan indtaste addreser og blive sent derhen (Mindre fejl gør, at det ikke fungerer lige nu)

Udført for v0.9: _____

- QuedTrees
- Kan nu vise hele Danmark (Ca. 4 min fra OSM)
- Man bliver sent hen til sin slut adresse (Mindre fejl gør, at dette ikke fungerer lige nu)
- Adresse forslag bliver automatisk sat op i takt med at man indskriver dem (Er i separat branch lige nu)
- Ikoner + Gade navne er blevet disablet

Wishlist for features for v1.0: _____

- Få Coastlines til at fungerer Shortest Path
- Fastest Path
- Rutevejledning
- Forbedret struktur til addreser
- Få binær load og save til at fungerer ordentlig med ny struktur i programmet.

A.6.10 v1.0

Features i v1.0: _____

- Ved at klikke og holde musen inde på det tegnede kort, trække derhen hvor man vil og slippe igen kan man se forskellige steder på kortet.
- Ved hjælp af musehjulet kan man zoome ind og ud på kortet, dette kan også gøres på mousepad eller ved scrollle ind og ud.
- Man zoomer derind, hvor musen er. Dets længere man zoomer ind, desto mere vises der på skærmen.

- Man kan indtaste addreser og blive sent derhen, og vejnavne forslag bliver sat op mens man indtaster addreserne.
- Det er muligt, at save kortet binært og loade det ind igen.
- Plus programmet har Bornholm- kortet indeni sig, som den kan starte op indenfra programmet.

Udført for v1.0 - 3/5 - 2015: _____

- Hav omkring Danmark bliver tegnet.
- Binær load og save. Bedste tid for loading af hele DK er 17 sek.
- Man bliver sent hen til sin slut adresse.
- Adresse forslag bliver automatisk sat op i takt med at man indskriver dem. Viser distance i bunden af skærmen.
- KD-tree for adresse punkterne.
- Viser denAdresse, der er nærmest på musen (Er i branch for sig).

Wishlist for features for feature freeze: _____

- Addreser bliver overført, når man loader binært. Binary
- Search Trie istedet for HashMap
- Shortest route Fastest route
- Rutevejledning fra punkt A til B.
- Viser distance i bunden af skærmen (Pænnere sat op + Med antialising). Colorblind mode
- Sæt binær fil af hele Danmark ind i jar filen.

A.6.11 Feature Freeze

Features: _____

- Ved at klikke og holde musen inde på det tegnede kort, trække derhen hvor man vil og slippe igen kan man se forskellige steder på kortet.
- Ved hjælp af musehjulet kan man zoome ind og ud på kortet, dette kan også gøres på mousepad eller ved scrollle ind og ud.
- Man zoomer derind, hvor musen er. Dets længere man zoomer ind, desto mere vises der på skærmen.
- Musen afgiver desuden den vej, som er nærmest på musen. Plus en geografiske position.
- Man kan indtaste addreser og blive sent derhen, og vejnavne forslag bliver sat op mens man indtaster addreserne.

- Trykkes der delete, så fjernes hele linjen.
- Har man indtastet en adresse i fra og til felterne, så laver programmet en route derhen, som er baseret på,
- om man ønsker den korteste eller hurtigste route. Hertil kan man vælge om ens transportmidel er en bil, cykel eller gåben.
- Hvilket programmet tager med i udregningen, når den laver routen. Desuden kan man fravælge at køre på moterveje.
- Det er muligt, at save kortet binært og loade det ind igen. Plus programmet har Bornholm-kortet indeni sig, som den kan starte op indenfra programmet. Hvilket den også kan gøre med Danmarks kortet som bin fil.
- Udover dette, så har programmet en "Colorbind mode", der gør farvene og stregerne nemmere at se for farveblinde mennesker, og den har en distance bar i bunden af venstre hjørne, der afgiver afstanden på stregen i forhold til ens zoom niveau.

Udført 10/5 - 2015: _____

- Addreser bliver overført, når man loader binært.
- Autocompliction af addreserne.
- Musen position afgiver adresse.
- Error labels, når brugeren laver en fejl.
- Binary Search Trie fungerer med ny datastruktur.
- Map af Danmark binært, kan blive loadet indenfra programmet.
- Shortest route
- Fastest route
- Transport måde - compobox
- Ignoring af moterveje knap
- Rutevejledning fra punkt A til B med passende beskrivelser (højre, venstre, ligeud). Colorblind mode